Universidade de Aveiro
2009
Departamento de Electrónica, Telecomunicações e Informática

Miguel Carragozela Lima da Fonseca

Dicionário Científico – Terminológico da Língua Portuguesa *On-line*

Universidade de Aveiro
2009
Departamento de Electrónica, Telecomunicações e Informática

Miguel Carragozela Lima da Fonseca

Dicionário Científico - Terminológico da Língua Portuguesa *On-line*

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica dos Professores Doutor Joaquim Arnaldo Carvalho Martins, Professor Catedrático do Departamento de Electrónica, Telecomunicações e Informática, e Doutor João Manuel Nunes Torrão, Professor Catedrático do Departamento de Línguas e Culturas, ambos da Universidade de Aveiro.

Para os meus pais pela paciência que tiveram, e para a minha irmã pela correcção do Português.

o júri

presidente

Doutor Armando José Formoso de Pinho
Professor Associado com agregação da Universidade de Aveiro

Doutor Fernando Joaquim Lopes Moreira
Professor Associado no Departamento Inovação, Ciência e Tecnologia da Universidade Portucalense

Doutor Joaquim Arnaldo Carvalho Martins
Professor Catedrático da Universidade de Aveiro

Doutor João Manuel Nunes Torrão
Professor Catedrático da Universidade de Aveiro

agradecimentos

Ao orientador professor Dr. Joaquim Arnaldo Martins pelo apoio; aos co-orientadores Dr. João Manuel Torrão e Dr. Carlos Morais pela disponibilidade; aos meus amigos e família pela força dada para a conclusão desta tese

palavras-chave

Web, base de dados, internet, Framework .NET, ASP .NET AJAX, javascript, , integração, css, html, conhecimento, web services, terminologia científica, etimologia.

resumo

Nos tempos que correm a importância da utilização da Internet como uma plataforma privilegiada de auxílio no estudo e análise de qualquer tipo de informação é inegável.

Esta dissertação apresenta um sistema de informação centrado na Web, denominado DiCiTer – Dicionário Científico - Terminológico da Língua Portuguesa, com um aspecto moderno e seguindo uma arquitectura actual.

Este sistema apresenta-se como uma grande plataforma para o estudo e compreensão das etimologias (e história) da terminologia científica. Para além disso, o sistema permite a contribuição dos utilizadores na adição de novos termos científicos que irão aumentar ainda mais a capacidade que o sistema tem ao ser usado como ferramenta de estudo e análise.

keywords

Web, data base, internet, Framework .NET, ASP .NET AJAX, javascript, , integration, css, html, knowledge, web services, scientific terminology, etymology.

abstract

Nowadays, the use of Internet as a privileged support platform for study and analysis of any kind of information is without question of major importance.
The following dissertation presents a Web-based information system, the Diciter – Dicionário Científico - Terminológico da Língua Portuguesa (Scientific and Terminological Dictionary of Portuguese Language), with a modern layout and following a modern architecture. The system presents itself as an extensive platform for the study and comprehension of etymology (as well as history) of scientific terminology. Moreover, the system allows the user to add new scientific terms, which will increase and improve the system's capacity when used as a tool for study and analysis.

Índice

Índice de tabelas

Índice de Imagens

Lista de Acrónimos

ACID	Atomicidade, Consistência, Isolamento, Durabilidade
AJAX	Assynchronous Javascript and XML
API	Application Programming Interface
ASP	Active Server Pages
CSS	Cascade Style Sheet
DHTML	Dynamic HyperText Markup Language
DOM	Document Object Model
FCL	Framework Class Library
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDE	Integrated Developement Environment
Java EE	Java Platform, Enterprise Edition
JSON	JavaScript Object Notation
JSP	Java Server Pages
LINQ	Language Integrated Query
PHP	Hypertext Preprocessor
RDMSs	Relational Database Management Systems
SGBD	Sistema de Gestão de Bases de Dados
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocal
UDDI	Universal Description, Discovery and Integration
URL	Uniform Resource Locator
WPF	Windows Presentation Form
WSDL	Web Services Description Language
XML	eXtensible Markup Language

1. Introdução

1.1. Enquadramento

Actualmente existem um conjunto de tecnologias informáticas que facilitam uma grande parte das actividades desenvolvidas pelo ser humano. Através da utilização de várias tecnologias é possível criar sistemas de informação que contribuem para um melhor acesso a essa informação independentemente da sua localização física. Esta capacidade é de extrema importância, pois trás grandes melhorias na quantidade e especialmente na qualidade da informação, para além de criar uma plataforma que facilita e melhora a sua análise e estudo.

Existe um conjunto variado de tecnologias que permitem a construção de sistemas de informação que seguem vários modelos e fornecem uma disponibilização de informação de uma forma rápida e eficiente. Através de sistemas de gestão de base de dados, que centralizam a informação de uma forma estruturada e ordenada, e de tecnologias de desenvolvimento de sistemas de informação centrados na Web, é possível a construção de sistemas que permitem uma gestão avançada da informação em qualquer lugar. Com tecnologias como a Framework .NET [1]**,** que apresenta capacidades de acesso a sistemas de gestão de base de dados, como o Microsoft SQL Server [2], é possível a construção de aplicações Web que realizem a gestão de toda a informação presente numa base de dados. Aplicando estas tecnologias já existentes e amadurecidas pelo tempo, que apresentam níveis de estabilidade e documentação elevados, é possível a construção de sistemas de informação de grande fiabilidade. Juntando a estas tecnologias outras mais recentes (como é caso do AJAX [3] - Assynchronous Javascript and XML) mas que trazem grandes benefícios a nível do aspecto e da utilização, é possível a criação de sistemas de informação poderosos, com grande acessibilidade e funções.

É com base nos princípios apresentados anteriormente que esta dissertação assenta, apresentando um sistema de informação centrado na Web, denominado DiCiTer – Dicionário Cientifico - Terminológico da Língua Portuguesa que apresenta similaridades com um dicionário online (que se encontra localizado no url, na altura em que esta tese foi realizada em diciter.web.ua.pt). É um dicionário etimológico e histórico da terminologia científica de base greco-latina que irá servir como ferramenta de auxílio na análise dos termos científicos e das suas fontes, através da segmentação e da identificação do valor dos seus constituintes morfológicos (lexemas, sufixos e prefixos), promovendo o conhecimento das etimologias de cada termo. O DiCiTer pode ser ainda utilizado como ferramenta de estudo que contribui:

- Para o conhecimento das bases greco-latinas das terminologias científicas;
- Para a compreensão do significado das raízes greco-latinas e dos processos de composição do vocabulário especializado;
- Para a dedução do significado de palavras das nomenclaturas das diferentes áreas científicas e técnicas, a partir do conhecimento dos constituintes de origem greco-latina;
- Para uma melhor compreensão do léxico científico e técnico e o rigor terminológico, pelo conhecimento das etimologias;
- Para a formação e análise de neologismos científicos de base greco-latina;
- Para a compreensão e reconhecimento de tecnicismos de origem greco-latina em diversas línguas modernas.

1.2. Objectivos

O principal objectivo deste trabalho consiste na construção de um sistema de informação centrado na Web que forneça instrumentos que permitam uma utilização precisa e rigorosa de vocabulário científico especializado. O sistema desenvolvido fornece mecanismos do género de dicionário online, permitindo pesquisas de termos de diferentes áreas científicas. É um sistema de livre acesso que abrange um amplo grupo de áreas científicas que vão desde a Medicina até à Astronomia, passando pela Farmácia, Botânica, Zoologia, Geologia, Química, Bioquímica, Física, Geografia, entre outras.

Esta aplicação foi desenvolvida tendo em conta o público em geral, mas tem como alvo principal o facto de ser uma ferramenta essencial para estudantes e técnicos de tradução especializada bem como profissionais das áreas científicas abordadas.

Resumidamente, os principais objectivos da dissertação formam a construção de uma aplicação Web:

- Com um aspecto moderno e de fácil utilização e compreensão.
- Que permita pesquisa de termos e correspondente disponibilização da sua etimologia e história;
- Que apresenta a possibilidade de pesquisas cruzadas de lexemas, de prefixos, de sufixos comuns a termos das diferentes áreas científicas presentes.
- Que permita a possibilidade de contributos externos, ou seja, que a própria aplicação permita a inserção de informação na base de dados.

- Que apresenta uma plataforma que seja uma mais-valia no estudo e análise da terminologia científica.

1.3. Estrutura

Esta dissertação encontra-se dividida em quatro capítulos:

O capítulo um, onde já foi apresentada a introdução e referidos os objectivos para o desenvolvimento da dissertação.

O capítulo dois que está dividido em duas partes. A primeira parte incide nos trabalhos existentes na área da terminologia. São referidos vários dicionários online, o Dicciomed [4], o MorDebe [5] e o Dicionário da Língua Portuguesa Houaiss [6]; e um livro português escrito pela Doutora Maria Madalena Dias Marques Contente [7].

A segunda parte aborda as tecnologias que foram utilizadas no desenvolvimento da aplicação proposta, incidindo na possibilidade que apresentam ao trabalharem todas em conjunto para a criação de um sistema de informação centrado na Web. Nesta parte, também é mencionada a razão da escolha dessas tecnologias, comparando-as com outras existentes.

No capítulo três é apresentada a arquitectura do sistema proposto. Neste capítulo são definidos os requisitos e a informação que dá origem à estrutura e organização da mesma no sistema de gestão de base de dados. Da mesma forma, incide na construção da aplicação Web, apontando algumas características e funcionalidades particulares que o sistema apresenta.

Por fim, no capítulo final é feito um balanço final, apresentando as conclusões obtidas e indicando possíveis melhoramentos futuros.

2. Estado da arte

2.1. Dicionários

Dicionário online Dicciomed.*es*

O Dicciomed.es *[4]* é um dicionário online médico – biológico espanhol, criado e coordenado pelo professor Francisco Cortés Gabaudan da Universidade de Salamanca. O Dicciomed.es foi criado através de uma tecnologia de desenvolvimento de páginas Web *open-source*, o PHP [8], sendo utilizado para o armazenamento da informação, novamente uma tecnologia *open-source* de gestão de base de dados, o MySQL [9].

Este dicionário está totalmente dedicado a termos oriundos das áreas da medicina e da biologia, contendo mais de 5000 entradas. O Dicciomed é uma ferramenta útil não só para o estudo das etimologias dos termos científicos como para as pessoas das áreas científicas que ele aborda. Possibilita um estudo da etimologia, apresentando para cada termo, os respectivos constituintes morfológicos, como os sufixos e os lexemas (os prefixos não são abordados) e um estudo da história de um termo, com referência ao ano e século de aparecimento do termo, classificando-o de acordo com a época em que surgiu, ou seja, como antigos, medievais ou como neologismos modernos posteriores ao Renascimento. Este estudo é completado com a apresentação de informação do género e da classe gramatical da área científica a que pertence e da língua-base. São ainda disponibilizados dados relativos a cada um dos constituintes morfológicos suportados. Para cada lexema é apresentado o étimo correspondente e o radical (ou raiz) do étimo. É de realçar, na caracterização do étimo de cada lexema e dos sufixos, o suporte de caracteres gregos. Para além desta informação o Dicciomed ainda disponibiliza listas dos termos armazenados, ordenadas cronológica e alfabeticamente, por campo científico e língua de origem. Apresenta também listas dos sufixos e lexemas armazenados, ordenadas alfabeticamente com a indicação do aparecimento dos mesmos na formação dos termos armazenados no dicionário.

Por fim, é de notar que o dicionário não suporta a adição online de novos termos. A adição de novos termos é realizada pela equipa científica do Dicciomed, podendo ser sugeridos novos termos, sujeitos a validação. Novos termos a adicionar podem ser sugeridos por alunos de uma disciplina leccionada no departamento de Biologia da Universidade de Salamanca, por correio electrónico, ou de uma forma curiosa a partir de pesquisas infrutíferas.

Outros dicionários on-line:

Existem vários dicionários online, mas a maior parte deles não suporta a mesma informação que o dicionário proposto nesta dissertação.

O MorDebe [5] é uma base de dados de palavras em português que fornece informação lexical como a ortografia, a flexão e as relações morfológicas entre palavras. De realçar que nada é apresentado sobre a semântica e sobre a etimologia das palavras. A maioria das palavras existentes nesta base de dados pertencem ao português europeu, mas também se encontram palavras provenientes de outros países lusófonos como Brasil, Angola, Moçambique, entre outros.

Talvez o dicionário escrito na língua Portuguesa, que apresenta alguma da informação existente no sistema proposto nesta dissertação seja o Dicionário da Língua Portuguesa Houaiss [6]. Este dicionário elaborado pelo lexicógrafo brasileiro António Houaiss contém uma quantidade elevada de termos, sinónimos, antónimos e palavras arcaicas. Apresenta também as etimologias de cada palavra e abrange a terminologia actual das áreas tecnológicas e científicas. Este dicionário só se encontra acessível dentro da rede informática da Universidade de Aveiro, não podendo ser acedido a partir do exterior.

Dicionário de Medicina – Sinónimos e equivalentes

Este dicionário informático é proposto no livro *"Terminocriatividade sinonímia e equivalência interlinguística em medicina*" [7] escrito pela Senhora Doutora Maria Madalena Dias Marques Contente, professora auxiliar convidada do Departamento de Educação Médica (DEM), da Faculdade de Ciências Médicas e investigadora do Centro de Linguística da Universidade Nova de Lisboa [1].

Neste livro, a Doutora Madalena Contente, que conta com o apoio da linha de Investigação de Lexicologia, Lexicografia e Terminologia do Centro de Linguística da Universidade Nova de Lisboa, propõe um dicionário terminológico de medicina que pressupõe um sistema de conhecimento conceptual e terminológico, ou seja, um sistema de sinónimos e equivalentes linguísticos. Para além de propor um sistema informático, os objectivos deste livro são de um

[1] A senhora doutora Maria Madalena Contente é ainda professora Titular da Escola Secundária de Camões. Apresenta como habilitações literárias, um doutoramento em Linguística com especialidade em Lexicologia (2004), pela Faculdade de Ciências Sociais de Humanas, Universidade Nova de Lisboa e um Mestrado em Linguística, especialidade Lexicologia e Lexicografia (1993), pela mesma Universidade.

modo geral a análise da construção do conceito e do termo científico, a análise da sua formação numa perspectiva unilingue e a análise de processos de criação de sinónimos em terminologia.

O sistema informático denominado *"Dicionário de Medicina – Sinónimos e equivalentes"* é uma aplicação *desktop*, não estando disponível na Internet. Este sistema de uma forma simplificada têm como objectivo a concepção de uma ferramenta que permita a análise em várias línguas (Português, Inglês e Francês) dos sinónimos e dos equivalentes de cada termo científico (em ambos os casos são fornecidos os traços semântico-conceptuais e os marcadores linguísticos do termo). O sistema informático apresentado no livro permite pesquisas de sinónimos e de equivalentes de termos científicos (nas várias línguas suportadas). De notar que para cada uma destas pesquisas existe a possibilidade de visualizar os traços semântico-conceptuais e os marcadores linguísticos respectivos ao termo pesquisado.

Para terminar convém realçar que no livro não é especificada a forma como a aplicação informática apresentada foi implementada, por isso não é possível levantar considerações acerca das tecnologias utilizadas nem da arquitectura apresentada.

2.2. Tecnologias

2.2.1. NET Framework

A Framework .NET [1] (a versão mais actual é a Framework .NET 3.5) é uma arquitectura e um conjunto de ferramentas criadas pela Microsoft, que possibilitam o desenvolvimento de aplicações. Esta ferramenta suporta a construção e a execução de aplicações de nova geração e de serviços Web, disponibilizando um conjunto de várias linguagens de programação.

A Framework .NET é um novo modelo de desenvolvimento de aplicações da Microsoft, não só para o sistema operativo Windows, mas também para outros sistemas operativos como o Mac OS e diferentes distribuições de Unix/Linux, através da utilização de uma Framework *open-source*, denominada Mono [10]. Algumas das características mais relevantes são a interoperabilidade com o código existente, a integração total e completa da linguagem, permitindo ao .NET suportar, por exemplo, a herança de classes, a gestão de excepções e o *debugging* entre o código construído com linguagens de programação diferentes, e a presença de um único motor de execução, partilhado por todas as linguagens. NET, e uma biblioteca de classes base abrangente.

A Framework .NET foi concebida tendo em vista os seguintes objectivos [1]:

- Proporcionar um ambiente de programação baseado no paradigma de programação e orientado aos objectos, em qualquer tipo de aplicação (para o código executado localmente; executado localmente mas distribuído pela Internet ou executado remotamente).
- Oferecer um ambiente de execução de código que minimize a distribuição do *software* e conflitos entre versões; que promova a execução segura do código, mesmo aquele criado por desconhecidos ou por alguém não fiável.
- Desenvolver uma variedade de tipos de aplicações, como as aplicações para Windows e as aplicações baseadas em Web, de forma consistente.
- Construir todo o tipo de comunicações com padrões da indústria, de forma a assegurar que o código baseado em .NET possa ser integrado com qualquer outro código.

A Framework .NET apresenta, na sua construção, três blocos principais, responsáveis pelas características mais importantes; o *common language runtime* (CLR), o *common type system* (CTS) e o *common language specification* (CLS), que são explicados a seguir.

Common Language Runtime – CLR:

O CLR é responsável pela gestão da execução do código .NET, incluindo a alocação de memória e respectiva limpeza. Também gere aspectos relacionados com segurança e gestão de *threads*.

Funciona com todas as linguagens que a Framework .NET disponibiliza, por isso não há necessidade de existir um ambiente de execução diferente para cada um das linguagens existentes.

Common Type System – CTS:

O CTS descreve todos os tipos de dados que são suportados pelo CLR, especificando como ocorre a interacção entre eles e como são representados no .NET.

Common Language Specification – CLS:

Nem todas as linguagens .NET podem suportar as características disponibilizadas pelo CTS. No entanto, o CLS define um conjunto de tipos e regras de programação que todas as linguagens .NET devem seguir.

O CLS tem como objectivo, possibilitar a interoperabilidade entre as linguagens .NET, incluindo a possibilidade de herança entre classes, escritas em linguagens .NET

diferentes. Por exemplo, se um programa aproveita, na totalidade, as características do CLS, a interoperabilidade entre esse programa e outros, escritos numa linguagem .NET, está assegurada.

Juntamente com os blocos principais mencionados, a Framework .NET apresenta uma biblioteca baseada em classes (FCL), disponível para todos os seus componentes, dos quais se destacam o ADO.NET, o ASP.NET, o Windows Forms e o Windows Presentation Foundation (WPF). Esta biblioteca de classes proporciona um suporte para um conjunto de serviços que são utilizados na maior parte das aplicações actuais. Esta biblioteca possui funções que facilitam o acesso e a manipulação da base de dados, a manipulação de documentos XML e a construção de aplicações Web.

Do mesmo modo, a Framework .NET apresenta diferentes linguagens de programação, das quais se destaca o C#. O C# é uma linguagem de programação elaborada pela Microsoft, especialmente para a plataforma .NET [11], e para o Common Language Infraestructure (CLI). É uma linguagem de programação multi-paradigma que engloba disciplinas de programação imperativa, funcional, genérica, orientadas aos objectos (baseada em classes). O C# tem como objectivo ser uma linguagem simples, segura, moderna e centrada na Internet. Esta linguagem apresenta uma capacidade de implementação de conceitos de programação modernos, numa linguagem estruturada e baseada em componentes, seguindo o paradigma da programação orientada aos objectos.

Em qualquer linguagem baseada no paradigma acima referido, é apresentado o suporte para a definição e programação com classes. O C# apresenta a possibilidade de declarar classes bem como os respectivos métodos e propriedades, podendo usar mecanismos de encapsulamento, de herança e polimorfismo, ou seja, os pilares da programação orientada aos objectos. O C#, sendo uma linguagem recente, é considerado como uma mistura de várias linguagens, tais como o Java, o C++ e o Visual Basic. Isto resulta numa linguagem mais limpa do que o Java, simples como o Visual Basic 6 e flexível como o C++. Segue uma lista das características mais importantes desta linguagem [11]:

- Não são necessários a utilização de apontadores;
- A gestão de memória automática é feita através de *garbage colection* (a memória utilizada é libertada automaticamente), não sendo necessário ao programador libertar explicitamente a memória usada, evitando assim falhas de memória.

- A existência de construtores para classes, interfaces, estruturas de dados, numerações, entre outros.
- A possibilidade de redefinir (*overload*) operadores para um tipo de dados personalizado, tal como o C++.
- Um suporte para programação baseada em atributos.
- Apresentação de suporte para internalização;
- Apresentação de suporte total para LINQ (*Language Intregrated Query*) – consulta (*queries*) de dados como parte da linguagem;
- Apresentação de suporte do Windows Presentation Foundation (WPF) para a criação de aplicações Windows; Work Flow (WF) e Silverlight (criação de aplicações Internet para vários *browsers* e plataformas).

2.2.2. ASP.NET

Desenvolvido pela Microsoft, a primeira versão do ASP.NET 1.0 saiu no dia 5 de Janeiro de 2002 como parte integrante da versão 1.0 da Framework .NET. Actualmente o ASP .NET encontra-se na versão 3.5, como parte integrante da Framework .NET com a mesma versão.

O ASP.NET [12] tem acesso a todos os conteúdos da Framework .NET, sendo uma tecnologia que permite a criação de aplicações Web interactivas e dinâmicas, centradas nos dados e executadas sobre a Internet e Intranets. O ASP .NET apresenta uma grande quantidade de controlos pré-construídos que podem ser configurados e manipulados através de código, para gerar páginas HTML, suportadas pelos *browsers* actuais. As páginas desenvolvidas através do ASP .NET são dinâmicas, porque ao contrário do HTML estático em que as páginas são totalmente armazenadas em disco, no ASP .NET são apenas armazenados esqueletos da página. Quando ocorre um pedido de uma pagina construída em ASP .NET, o código ASP é processado no servidor, gerando o código HTML da página, que será enviado para o *browser*.

Como parte integrante da Framework .NET, o ASP .NET herda as características da Framework, sendo uma das mais importantes, a utilização do paradigma de programação orientada aos objectos. O ASP.NET trouxe esse paradigma para a construção de aplicações Web, possibilitando a utilização do modelo de *code behind*. Este modelo marca a separação do ASP clássico, permitindo aos programadores desenvolverem o código executável separado da interface do utilizador (do aspecto da página). Assim surge o conceito de *Web forms*, considerado para muitos como os blocos de construção principal do ASP .NET para a *Web* [13].

Os *Web forms* no ASP .NET oferecem algumas vantagens, quando comparados com outras tecnologias de criação de aplicações Web. Algumas das suas características principais são:

- O suporte para qualquer *browser,* compatível com o HTML 3.2.. De realçar os controlos do servidor do ASP .NET que proporcionam funcionalidades avançadas ao cliente e que podem diminuir essas funcionalidades para *browsers* que não suportem DHTML (*Dynamic* HTML) ou *scripts*.
- A construção no conceito *Common Language Runtime* (CLR), beneficiando de todos os seus recursos, como a gestão da execução e a herança.
- A construção em qualquer linguagem baseada no CLR, como o C#, Visual Basic .NET, JScript .NET.
- O desenvolvimento rápido, utilizando ferramentas como o Visual Studio.
- A apresentação de um conjunto de controlos de servidor que proporcionam quase todas as funcionalidades necessárias para a construção de aplicações Web.
- A preservação do estado da página e dos seus controlos entre pedidos, com a inclusão de funcionalidades de gestão de estado.
- A apresentação de um modelo extensível que permite o desenvolvimento de controlos próprios e a utilização de controlos de terceiros.

Talvez a característica mais importante dos *Web forms* seja a implementação de um modelo de programação, em que as páginas são geradas dinamicamente num servidor Web, antes de serem enviadas para o *browser*. Com os *Web forms* é criada uma página aspx onde é colocado o conteúdo mais ou menos estático, consistindo em controlos HTML e controlos Web, bem como controlos AJAX e Silverlight [14]. Este ficheiro tem o nome da interface do utilizador, uma vez que é aí definido o aspecto e organização da página. O código que irá adicionar o conteúdo dinâmico é colocado num ficheiro, que apresenta o mesmo nome do ficheiro da interface do utilizador, mas com extensão .aspx.cs ou aspx.vb, dependendo se a linguagem de programação utilizada é o C# ou o Visual Basic .NET, respectivamente. Este código vai ser executado no servidor para os eventos do aspx e o resultado que for produzido é integrado com o interface do utilizador, de forma a criar uma página HTML que é enviada para o *browser*.

Assim o *Web forms* divide a interface em duas partes: uma parte visual, e uma parte lógica que completa a parte visual.

Uma outra característica que vale a pena referir no ASP.NET é a possibilidade da utilização das *Master Pages*. É possível considerar as *Master Pages* como templates, que possibilitam a um conteúdo comum existente num site Web ser partilhado entre as várias páginas que o

constituem, sem que haja a necessidade de repetir esse conteúdo em cada página. Assim uma página usa o seu conteúdo e o da *master page* para criar o seu aspecto final. De notar que a *master page* fica totalmente acessível para a página que a utiliza, podendo manipular os cabeçalhos, mudar o título e aceder a propriedades e métodos públicos presentes na *master page*.

2.2.3. *Web Services*

O objectivo principal dos *Web Services [15]* do ASP .NET é fornecer acesso a funcionalidades de aplicações, disponibilizadas através de protocolos padrão da Web, como é o caso do HTML e do XML, independentemente da sua localização. Para que o acesso aos serviços seja possível, apesar das plataformas envolvidas, quer no serviço, quer no cliente, é necessário que exista um suporte para XML e SOAP, e que seja possível comunicar através do protocolo HTTP. De uma forma geral, é possível afirmar que um *Web Service* é uma função invocada através da Internet [13]. Essa funcionalidade disponibilizada por um *Web Service* tanto pode ser de baixa, ou de alta complexidade, podendo enviar ao cliente vários tipos de dados, como inteiros, *strings* simples, e tipos de dados complexos como classes e mesmo partes de código HTML.

A capacidade da interoperabilidade entre aplicações e a respectiva integração que os *Web Services* apresentam, depende dos seguintes de padrões da Web:

HTTP [16]:

O http é o protocolo padrão para a World Wide Web. É essencial para os *Web services*, devido ao facto de a porta utilizada pelo http (a porta *tcp* 80) não se encontrar normalmente bloqueada nas *firewalls*, possibilitando assim que os *Web services* comuniquem livremente na Internet.

XML [17]:

O XML, ou Extensible Markup Language, é uma forma padrão de estruturação de dados, através de uma sintaxe baseada em *tags* e texto, que quando combinada com o padrão XML Schema [18] tem a capacidade de converter e de voltar à forma original, certos tipos de dados simples e complexos para texto, de forma a serem transmitidos por http.

SOAP [19]

O Simple Object Access Protocol é um protocolo que especifica o formato das mensagens de pedidos e respostas, usando XML sobre http. É essencial para os *Web Services*, porque é responsável pelo transporte dos dados de e para os *Web Services*. Este protocolo apresenta uma característica muito importante de não especificação, e de não definição do modelo de programação ou de implementação para os serviços, garantindo assim que o serviço e o cliente que o invocam, possam ser desenvolvidos em linguagens de programação diferentes.

WSDL [20]:

O Web Services Description Language (WSDL) é utilizado para a descrição dos *Web Services*, contendo informação relativa a operações, mensagens e parâmetros. O WSDL proporciona assim uma forma de anunciar os métodos disponíveis de um *Web Service*. É a partir do WSDL que os clientes, que desejam utilizar um *Web Service*, descobrem como devem invocar o serviço para que funcione correctamente. O WSDL é responsável pela descrição dos serviços, enquanto o SOAP é responsável pela comunicação entre o cliente e o servidor.

UDDI [21]:

O Universal Description, Discovery, and Integration é uma Framework que não depende de nenhuma plataforma, que se encontra em desenvolvimento pela Microsoft e pela IBM, que oferece uma forma de publicação, localização e integração/ligação a *Web Services*. Os *Web Services* podem ser registados num dos directórios do UDDI, para que os clientes interessados nos serviços os possam pesquisar e aceder aos que interessam.

2.2.4. JavaScript

O JavaScript (ECMAScript) [22] é uma das mais utilizadas linguagens de programação. Esta linguagem teve um grande desenvolvimento nos últimos anos, por ser utilizada na maior parte dos *Web sites*.

Apesar de muitas vezes ser considerado uma linguagem de programação leve e pouco sofisticada [23], o JavaScript é uma linguagem totalmente integrada nos *Web browsers*, que possibilita a criação de efeitos nas páginas Web. Esta linguagem possibilita, por exemplo, a criação de nova janela ou *popup*, sendo possível controlar o tamanho, a posição e outros atributos dessa janela (apresentação de menus e barra de ferramentas). Da mesma forma, possibilita também a

validação dos valores de *input* de um formulário *Web* (de forma a corrigir potenciais erros, antes de serem enviados para o servidor) e a mudança de imagens quando sobrepostas pelo cursor do rato, para chamar a atenção do utilizador. O JavaScript pode detectar *keystrokes* individuais, algo que o HTML não consegue fazer. Esta funcionalidade permite, por exemplo, a definição de um teclado em grego, que suporta todos os acentos e caracteres especiais.

O JavaScript é implementado dentro de um ambiente específico, normalmente um *Web browser*. Como medida de segurança, corre num ambiente fechado denominado de *sand-box*. Neste ambiente, o *script* só pode realizar operações que estejam relacionadas com o visionamento de páginas, não podendo realizar tarefas globais como a criação de um ficheiro. Outra medida de segurança é o respeito de uma política da mesma origem, ou seja, os scripts de um *Web site* não tem acesso a informação, como o nome dos utilizadores, as palavras-chave ou os *cookies* enviados para outro *Web site*.

No entanto, apresenta alguns problemas, geradores de conflitos entre utilizadores, podendo criar problemas de segurança se for utilizado incorrectamente, especialmente quando combinado com outras funcionalidades, como um *Web service* e uma base de dados. Da mesma forma, pode tornar uma página inutilizável, impossível de ler e com menor acessibilidade.

A linguagem JavaScript corre numa grande variedade de ambientes e de plataformas, e pode ser utilizada no desenvolvimento de páginas Web que correm em sistemas operativos como o Mac OS X, Windows e Linux, não necessitando de nenhuma aplicação especial, uma vez que está embutida no *browser*. A maior parte dos *browsers* implementa um subconjunto comum desta linguagem, tornando compatível a maior parte do código JavaScript. A maior parte das incompatibilidades existentes tem origem nas diferentes formas como o *browser* trata o Document Object Model (DOM) [24]. O DOM é uma especificação da W3C [25], independente da plataforma e da linguagem, que permite a representação e interacção com objectos dentro de documentos HTML, XHTML e XML. O DOM é a forma como o JavaScript analisa o conteúdo de uma página HTML e o estado do *browser*, sendo utilizado por *scripts* JavaScript que necessitam de aceder ou modificar, de forma dinâmica, uma página Web.

É devido à forma como os objectos DOM são manipulados, através da implementação do *browser* do JavaScript, que se originam as incompatibilidades. Essas incompatibilidades são provocadas por implementações diferentes que cada *browser* apresenta para o modelo DOM. Para resolver essas incompatibilidades é necessário que o código seja compatível com a implementação dos objectos DOM e isso só é possível com um código que detecte o tipo de

browser e funcione de acordo com as características da implementação do DOM, apresentada pelo *browser*.

2.2.5. CSS

Enquanto o JavaScript tem a responsabilidade de criar, remover ou alterar os atributos de um elemento de uma página, o CSS [26] é responsável por definir esses atributos, através dos estilos CSS.

O CSS define o aspecto e o comportamento de elementos presentes numa página Web, permitindo mostrar ou esconder elementos, alterar a cor, o tipo de letra, mover ou redimensionar. A forma como cada *browser* gere estas funcionalidades, pode, uma vez mais, ser diferente de *browser* para *browser*, possibilitando o aparecimento de incompatibilidades.

2.2.6. ASP.NET AJAX

O ASP .NET AJAX [3] (Asynchronous JavaScript e XML) é um conjunto de tecnologias (da Microsoft) que se relacionam entre si, e quando utilizadas num sistema cliente-servidor possibilitam a criação de aplicações Web interactivas mais apelativas e com melhor aspecto, e que respondem mais rapidamente a pedidos do utilizador.

A funcionalidade do ASP.NET AJAX assenta em tecnologias que estão presentes nos *browsers* actuais: o JavaScript assíncrono e o XML. É graças a estas tecnologias que o AJAX apresenta a capacidade de desenvolver as páginas Web que permitem realizar pedidos http em plano de fundo, ou de forma assíncrona, sem que para tal seja necessário reler a totalidade da página, tornando mais rápida a resposta a um pedido, ou que esta pareça mais rápida ao utilizador. Isto é possível devido à utilização de tecnologias, por parte do AJAX, que são suportadas pelos *browsers* actuais e das quais se destacam: o JavaScript, o Document Object Model (DOM) [24] e o Cascading Style Sheets (CSS) [27].

A Framework ASP .NET AJAX inclui bibliotecas que incluem *scripts* para o cliente e que providenciam um conjunto de vantagens para o programador. Essas vantagens são [27]:

Camada de compatibilidade com o browser:

Possibilita aos elementos ASP .NET AJAX correrem na maior parte dos *browsers* actuais, quase eliminando a necessidade de criar *scripts* específicos para cada *browser* existente.

Core services:

Torna possível escrever *scripts* JavaScript de uma forma semelhante ao paradigma de programação orientada aos objectos. Isto inclui o suporte para classes, *namespaces*, gestão de eventos, herança, e serialização de objectos, através do formato JSON (JavaScript Object Notation) e XML.

Biblioteca baseada em classes:

Esta biblioteca contém componentes do estilo .NET, como construtores de *strings* e temporizadores.

Componentes e controlo de script:

Fornece ao ASP.NET AJAX versões dos controlos em HTML padrão, podendo acrescentar-se outras funcionalidades como ligação a dados, comportamentos pré-definidos (como é a funcionalidade de *Drag-and-Drop*). Estes controlos podem ser programados directamente ou através da utilização de script XML.

Tal como o ASP .NET, o ASP.NET AJAX foi concebido para que as suas funcionalidades possam ser utilizadas sem ser necessário o domínio das tecnologias presentes no AJAX. Essas funcionalidades são geridas da mesma forma que o ASP .NET gere as funcionalidades do http. O objectivo do ASP .NET AJAX é facilitar o trabalho do programador a dois níveis: ao nível do cliente, oferecendo funções JavaScript para o envio dos pedidos ao servidor e ao nível do servidor, trabalhando em conjunto com o ASP.NET, o que permite aos controlos do ASP.NET AJAX interagir com controlos e componentes ASP .NET, e participar no ciclo de vida da página. O ASP .NET AJAX pode ser adicionado a características do ASP .NET tais como sessões, autenticação e perfis de utilizador.

Estão disponíveis diferentes pacotes do ASP .NET AJAX que apresentam diferentes funcionalidades, sendo o mais relevante o ASP.NET AJAX Control Toolkit. Este pacote, disponibilizado em *open-source*, contém um conjunto de componentes que permite utilizar facilmente as funcionalidades do AJAX, com controlos reutilizáveis e personalizáveis que podem ser utilizados na criação de páginas Web dinâmicas e interactivas.

2.2.7. SQL Server

O Microsoft SQL Server [2] é um Sistema de Gestão de Base de Dados - SGBD, em que as respectivas bases de dados (tabelas), nele armazenadas, seguem o modelo relacional. A versão utilizada neste projecto foi o Microsoft SQL Server 2005, apesar de já existir uma versão mais recente (2008) que no início do projecto ainda não estava disponível.

O modelo relacional segue um conjunto de princípios matemáticos oriundos da teoria dos conjuntos e da lógica. Este modelo define a forma como os dados podem ser protegidos (integridade dos dados) e as operações que podem ser efectuadas sobre esses dados. Devido à sua flexibilidade e eficiência, o modelo relacional é o mais usado em sistemas de base de dados. Com este modelo é possível realizar alterações à estrutura da base de dados, sem que sejam necessárias alterações, numa aplicação que utilize essa mesma base de dados. Uma vez que o modelo apresenta uma flexibilidade estrutural, é possível a recuperação de combinações dos dados que não foram pensadas durante a concepção da base de dados.

Os sistemas de base de dados relacionais possuem as seguintes características gerais:

- Todos os dados são representados como um arranjo de linhas e colunas, denominado relação;
- Todos os valores são escalares, dada uma qualquer linha/coluna e a relação aí existente é só um e um só valor;
- Todas as operações são realizadas na relação e os resultados obtidos são representados também como relações.

Sendo então o Microsoft SQL Server um sistema de gestão de base de dados relacional - Relational Database Management Systems (RDBMSs), os dados estão armazenados em tabelas, compostas por linhas e colunas. As tabelas com dados independentes podem ser ligadas ou relacionadas umas às outras.

O SQL Server 2005 apresenta algumas características importantes que devem ser analisadas:

- Capacidade de utilização do *common language runtime* (CLR) da Framework .NET na base de dados. Assim é possível utilizar as linguagens de programação Visual Basic ou C# na base de dados;
- Suporte para XML através de um tipo de dados XML que apresenta todas as capacidades dos dados relacionais, possibilitando a inserção de um ficheiro XML na base de dados, validando o documento, e a extracção de apenas partes do documento quando necessário.

- Novo interface de gestão denominado de SQL *Server Management Studio*, que permite num só sítio a gestão e administração do servidor;
- Framework de *reporting* (SQL Server Reporting Services) como parte integrante do sistema de base de dados;
- Nova aplicação de *Service Broker*, para a entrega de mensagens de forma assíncrona;
- Ferramenta melhorada que permite a extracção, transformação e leitura de dados – SQL Server Integration Services.

O SQL Server apresenta como linguagem principal de programação e gestão, o T-SQL (*transact* – SQL). Esta apresenta palavras-chave e/ou funções que permitem criar e alterar as bases de dados, bem como monitorização e gestão do próprio servidor. Esta capacidade de gestão só é possível graças à existência de tabelas e de *stored procedures* de sistema que podem ser invocados através de *queries* T-SQL.

O SQL Server para além de apresentar a linguagem de programação T-SQL, apresenta algumas funcionalidades baseadas nessa linguagem, consideradas importantes. Essas funcionalidades são transacções, restrições (*constrains*), índices, *Views*, *triggers* e *stored procedures* [28].

Transacções:

O SQL server assegura que toda e qualquer alteração nos dados respeitam o conceito ACID (Atomicidade, Consistência, Isolamento, Durabilidade). O ACID é um conjunto de propriedades que garante uma execução correcta das transacções numa base de dados.

Como mencionado anteriormente, o conceito ACID é um acrónimo de Atomicidade, Consistência, Isolamento e Durabilidade. Cada termo define uma propriedade importante no funcionamento não só do SQL Server, mas também dos SGBDs em geral:

Atomicidade:

Todas as operações presentes numa transacção devem ser executadas correctamente ou, em caso de alguma operação falhar, o resultado de todas as acções, pertencentes a uma mesma transacção que já tenham sido executadas, não é armazenado na base de dados.

Consistência:

A base de dados deve ser mantida num estado de consistência de informação antes do início e depois do fim de uma transacção, sendo esta realizada com sucesso ou não.

Só informação válida é que pode ser escrita na base de dados. Se por alguma razão, a transacção executada violar as regras de consistência da base de dados, esta é revertida e a consistência dos dados restabelecida.

Isolamento:

As operações não podem aceder nem "ver" a informação num estado intermédio durante uma transacção, ou seja, uma transacção não tem conhecimento de outras transacções em execução concorrente no sistema.

Durabilidade:

Depois da conclusão de uma transacção, armazenamento e notificação com sucesso, a alteração dos dados vai ser mantida, mesmo que ocorra uma falha no sistema. Muitas bases de dados implementam a durabilidade através de um *log* de transacções onde são armazenadas todas as transacções. As transacções só são realizadas depois de terem sido escritas no *log*. No caso da ocorrência de uma falha, o *log* pode ser lido, de forma a colocar o sistema no estado em que estava antes da falha.

Resumindo, as transacções possibilitam a definição de uma acção considerada atómica. Além disso, permitem definir um conjunto de acções que vão ser consideradas como atómicas. O comportamento padrão do SQL é considerar qualquer operação como transacção, caso não apareça dentro um bloco BEGIN TRAN/COMMIT TRAN. Se for necessário declarar um conjunto de operações como transacções, basta colocá-las dentro desse bloco .

Índices

Um índice numa tabela ou numa coluna de uma base de dados possibilita uma pesquisa mais rápida, quando comparada com a pesquisa numa tabela completa, porque o índice contém uma lista ordenada de elementos. Os índices podem contribuir para um melhor desempenho das *queries* executadas, mas a criação de um conjunto de *índices* nem sempre é fácil e directa.

Um índice bem construído reduz as operações de *input* e *output* no disco, sendo estas operações de certo modo demoradas e com efeitos negativos na performance. Quanto menor for

o tempo de acesso, melhor será a performance da aplicação. Isto não quer dizer que quanto maior for o número de índices que uma tabela possui, melhor será a performance. É necessário ter em conta que, em cada actualização dos dados da tabela, o índice também será actualizado, provocando uma degradação na performance.

O SQL Server 2005 suporta vários tipos de índice:

- ***Clustered***: Cria um índice para todas as colunas de uma tabela;
- ***NonClustered***: Indexa uma ou mais colunas de uma tabela;
- ***Unique***: Cria um índice que impossibilita a existência de valores repetidos na coluna que está a ser indexada.
- ***Vistas Indexadas***: As vistas, tal como as tabelas, podem conter índices;

Restrições:

O objectivo das restrições é restringir os valores que uma coluna pode ter. As restrições asseguram a integridade dos dados. Em geral, podem ser associadas a uma tabela como um todo ou a uma coluna. Uma restrição pode ser de diferentes tipos, cada uma com um funcionamento distinto:

- *not null* – especifica que a coluna não pode conter valores nulos.
- *unique* – especifica que o valor da coluna tem de ser único ao longo de todas as linhas de uma tabela.
- *primary key* – especifica um identificador único para a tabela (pode ser uma coluna ou um conjunto)
- *references* – é utilizada para a criação de chaves secundárias, que indicam a existência de uma relação entre várias tabelas. Esta chave refere a chave primária de outra tabela que participa na relação.
- *check* – especifica uma condição que a coluna tem de satisfazer.

Views:

Uma *View* é uma tabela virtual construída por uma *query* que pode ser utilizada como uma tabela normal.

Apesar de uma *View* poder ser constituída por dados de várias tabelas, o que obriga o SQL Server a percorrer todas as tabelas para preencher a *View*, não apresenta um impacto negativo

nem positivo em termos de performance. Uma forma de melhorar o desempenho das *Views* é a adição de índices.

As *Views* podem ser criadas por razões de segurança, quando não é desejável apresentar toda a informação, exibindo apenas uma parte dos dados. Também podem ser utilizadas para desenvolver soluções simples de uma forma modular. Como funcionam como tabelas, é possível dividir a informação a pesquisar por diferentes *Views* e posteriormente juntá-las, para obter a totalidade da informação desejada. Assim a construção da pesquisa torna-se mais fácil e de melhor compreensão.

Stored procedures:

Os *stored procedures* são rotinas executáveis no servidor de gestão de base de dados. Têm a capacidade de alterar os dados das tabelas. É possível controlar aos dados, garantido permissões aos *stored procedures*. De realçar que a possibilidade de aceitarem parâmetros de entrada e de saída, podendo ser validados antes realizar alguma operação complexa.

A utilização do *stored procedure* é vantajosa, porque além de evitar a repetição de código, quando é executado, fica armazenado em *cache*, tornando os acessos futuros mais rápidos e salvando os recursos do servidor. Graças a utilização de *stored procedures* o tráfego de rede é reduzido, porque só é necessário especificar o nome e os argumentos para os invocar em vez do código completo, necessário para a operação. Toda a actividade de processamento é realizada no servidor e só o resultando final é enviado, através da rede para o cliente.

***Triggers*:**

Os *triggers* são rotinas que se executam automaticamente como resultado da ocorrência de um evento pré-definido. No SQL Server existem dois tipos de *triggers*:

- *After triggers*: são executados em reacção automática a *queries* do utilizador ou a aplicações. Podem ser utilizados para reagir às acções de CREATE, ALTER e DROP;

- *Instead of triggers*: são executados como substituição de uma *query*. Podem ser utilizados para substituir acções de INSERT, UPDATE e DELETE.

Os *triggers* embora não possuam parâmetros de entrada e saída, não podem ser invocados directamente, estando a sua execução dependente de uma acção específica provocada por uma *query*. Assim permitem a automatização de processos, o que significa menos trabalho

manual e menor probabilidade de ocorrência de erros, sendo úteis na verificação da integridade dos dados.

2.2.8. Microsoft Visual Studio 2008

O Microsoft Visual Studio 2008 [29] é um IDE (Integrated Developement Environment), um conjunto de ferramentas que permite aos programadores criarem aplicações na plataforma Windows. Este conjunto de ferramentas permite o desenvolvimento de aplicações ASP .NET para Web, Web Services XML, aplicações de *desktop* e telemóveis. Todas as linguagens presentes na Framework .NET (Visual Basic, Visual C++, Visual C# e Visual J#) utilizam o este IDE, o que permite partilhar funcionalidades e criar soluções como a utilização das várias linguagens.

Para o desenvolvimento de aplicações, o Microsoft Visual Studio recorre à Framework .NET e a todas as suas funcionalidades como as linguagens Visual Basic e o C#, e à plataforma ASP .NET, para o desenvolvimento de aplicações para Web. Na sua versão actual, o Visual Studio 2008 apresenta um suporte para o Windows Vista, para o ASP .NET AJAX e para o Silverlight.

O Visual Studio apresenta uma interface de desenvolvimento visual (Visual Web Developer), que permite a criação e edição de páginas Web ASP .NET e páginas HTML, possibilitando uma forma simples e rápida de criar Web Forms. Também suporta todas funcionalidades do ASP .NET, permitindo um desenvolvimento rápido e eficaz de aplicações Web. Com o Visual Studio é possível a criação e gestão de *Web sites*, colocados em directórios locais, no servidor Web do Windows (o Internet Information Services - ISS), ou num servidor FTP

2.2.9 Outras tecnologias

2.2.9.1. PHP

O PHP (Hypertext Preprocessor) [8] é uma linguagem de *script open-source* largamente utilizada. Foi especialmente desenhada para o desenvolvimento Web, podendo ser utilizada juntamente com HTML.

As páginas desenvolvidas com PHP apresentam código HTML bem como código PHP, que realiza funções que aumentam as funcionalidades do HTML. Como acontece no ASP. NET, esse código é executado no servidor que gera o código HTML, sendo posteriormente enviado para o *browser* e permitindo a visualização da página. No entanto, não é possível visualizar no *browser* do cliente, o código PHP que gerou essa página.

O PHP apresenta algumas desvantagens em relação ao ASP.NET. Em primeiro lugar, o ASP .NET segue o paradigma de programação orientado aos objectos e por isso, tira proveito das funcionalidades apresentadas pelo CLR (common language runtime), que permite a utilização das várias linguagens disponíveis na Framework .NET. Em segundo lugar, o ASP . NET apresenta um ambiente de desenvolvimento totalmente integrado com a Framework .NET e com sistemas de base de dados (Microsoft SQL Server). Esta integração oferece ao ASP .NET uma facilidade de utilização e de construção tanto em aplicações Web simples ou mais complexas, como a utilização de Base de Dados.

No entanto, o PHP revela algumas vantagens como a distribuição gratuita, o que já não acontece no ASP .NET, uma vez que versões mais poderosas e com maior número de funcionalidades têm de ser adquiridas. Uma outra vantagem do PHP é o seu carácter multi-plataforma, com várias ferramentas necessárias para o seu funcionamento, desde servidores Web, sistemas de base de dados com suporte para várias plataformas e sistemas operativos.

2.2.9.2. MySQL

O MySQL [9] é um sistema de gestão de base de dados em código aberto, suportado pela Sun Microsystems. O MySQL tem sido nos últimos tempos o SGBD preferido da comunidade *open-source*.

Nas versões anteriores à actual 5.X, o MySQL ainda não possuía suporte para funcionalidades importantes como cursores, vistas e *stored procedures*, funcionalidades que já são suportadas há muito tempo pelo SQL Server da Microsoft. Na versão 5.X essa lacuna foi corrigida, mas como a sua implementação é recente, as funcionalidades ainda não se encontram completamente implementadas e enraizadas como no Microsoft SQL Server.

O MySQL apresenta ainda diversas arquitecturas de base de dados, que apesar de oferecerem um maior leque de escolhas aos programadores, provocam complicações na implementação da replicação de servidores, processamento paralelo e recuperação entre bases de dados de arquitectura diferente.

Como referido anteriormente, o MySQL é distribuído em *open-source* e apesar do SQL Server, para o meio empresarial, ser uma solução paga, é a mais evoluída tecnologicamente para sistemas de base de dados com tamanho considerável. A sua grande integração com a Framework .NET da Microsoft, e com o Visual Studio, facilita a construção de aplicações sem haver necessidade de aprender funcionalidades avançadas da linguagem SQL.

Concluindo, é possível afirmar que o MySQL é uma solução para aplicações de pequena complexidade e com poucos requisitos de segurança e escalonamento. O MySQL pode fornecer as funcionalidades básicas de um SGBD a custos mais reduzidos. O SQL Server é mais indicado para aplicações empresariais em que requisitos mais avançados sejam importantes, como a replicação, *clustering*, segurança e utilização de ferramentas de gestão.

2.2.9.3. Java

O Java [30] é uma linguagem de programação poderosa, desenvolvida e distribuída pela Sun Microsystems. É disponibilizada em *open-source* o que possibilita a construção de aplicações Java quase sem custo, uma vez que não é necessário comprar licenças para a utilização nem para distribuição. Para além desta vantagem, uma outra grande vantagem do Java é o facto de poder ser executado em várias plataformas. Estão disponíveis versões para a maior parte dos sistemas operativos como Windows, Mac OS e todas as diferentes versões do Linux, permitindo todas elas a execução de programas implementados em Java bem como a construção dos mesmos. De notar que um dos objectivos do Java é permitir que um programa, que tenha sido implementado em qualquer um dos sistemas operativos mencionados, possa correr em qualquer um deles, ou seja, uma aplicação desenvolvida no Windows pode ser executada no Mac OS ou em Linux sem qualquer problema. É possível afirmar que este objectivo é cumprido de um modo geral, mas, apesar de ser raro, é possível que ocorram alguns problemas de portabilidade que não permitam a execução de uma dada aplicação.

O Java possui uma plataforma a Java Enterprise Edition (Java EE) [31], que tal como a Framework .NET da Microsoft, possibilita a implementação de aplicações dinâmicas centradas na Web, bem como a construção de serviços segundo a arquitectura SOA. O Java EE é um conjunto de tecnologias que reduzem a complexidade de desenvolvimento e gestão de aplicações Web com ou sem integração de dados. Para esse desenvolvimento de aplicações Web, o Java EE apresenta uma tecnologia denominada de JavaServer Pages (JSP) [32]. O JSP é uma tecnologia que possibilita a construção de páginas Web dinâmicas, através da utilização de HTML ou XML semelhante ao ASP da Microsoft e ao PHP. Como é uma tecnologia Java, funciona independentemente da plataforma utilizada, permitindo a sua execução em sistemas operacionais distintos. Tal como o ASP .NET, esta tecnologia permite o desenvolvimento de aplicações centradas na Web, separando o código que cria a interface do utilizador, do código que gera os conteúdos dinâmicos. Estas páginas podem aceder a sistemas de gestão de base de dados

de forma a integrarem esses dados na própria página. De realçar que o Java EE suporta uma grande variedade de sistemas de gestão de base de dados de onde se destacam o MySQL [9], o Apache Derby [33] e o Oracle [34].

Para a disponibilização de páginas JSP na Internet, é necessária a sua instalação num servidor Web compatível com a tecnologia Java EE como o Apache Tomcat [35] e o GlassFish [36]. São estes servidores que irão gerar o código HTML correspondente ao código JSP de cada página para que seja possível a sua visualização no *browser* do cliente.

Comparando as tecnologias Java EE e ASP .NET, a primeira comparação está relacionada com o custo monetário de cada tecnologia. Para um melhor proveito da plataforma. NET é necessário adquirir um servidor e ferramentas de desenvolvimento da Microsoft que não são disponibilizadas em *open-source*, apresentando assim um custo algo elevado. É verdade que existem versões gratuitas de ferramentas baseadas no ASP.NET, fornecidas até pela própria Microsoft ou por terceiros no caso do Mono, mas estas ferramentas não disponibilizam a totalidade das funções presentes nas versões pagas. Pelo contrário, o Java é disponibilizado em *open-source*, bem como a maior parte das suas tecnologias e ferramentas disponíveis para o desenvolvimento de páginas Web dinâmicas, como os servidores Web e os sistemas de gestão de base de dados. Uma outra comparação possível entre o Java e o ASP.NET é a capacidade que o Java tem para ser executado em diferentes sistemas operativos, sem prejuízo da performance, enquanto o ASP.NET necessita de ser executado num ambiente Windows, para melhor aproveitamento da totalidade das respectivas capacidades.

Talvez a maior vantagem apresentada pela Framework .NET , e por consequência pelo ASP .NET seja a possibilidade da utilização de várias línguas de programação numa mesma aplicação de um modo transparente e sem necessidade de reconfigurações por parte do programador. Esta capacidade permite que o desenvolvimento das aplicações não fique preso numa só linguagem, sendo de realçar que a Framework .NET inclui uma linguagem muito semelhante ao Java, o J#. Outra grande vantagem é a facilidade da troca de dados entre as diferentes aplicações, de e para um sistema de gestão de base de dados. Como o Microsoft SQL Server e a Framework .NET são desenvolvidos pela Microsoft a comunicação entre estes dois sistemas é fácil e de rápida implementação.

Para finalizar, é necessário referir os IDE (*Integrated Development Environment)* ou em português - Ambiente Integrado de Desenvolvimento, existentes tanto para a Framework .NET como para o Java. O IDE da Microsoft, o Visual Studio, apresenta uma utilização e construção de aplicações Web mais fácil, pela seguinte razão: o código apresentado numa aplicação ASP.NET é

mais limpo e apresentável do que o código apresentado numa aplicação implementada com JSP. Para além disso, a Framework Java apresentou ao longo do tempo um crescimento considerável, o que a torna bastante pesada em termos de recursos computacionais. A diferença da velocidade de funcionamento que o IDE Visual Studio apresenta em situações análogas é tremenda, quando comparado com os seus homólogos para o Java, como por exemplo o NetBeans [37] ou o Eclipse [38], o que só trás vantagens no desenvolvimento de aplicações através do IDE Visual Studio e da tecnologia .NET Framework.

3. Arquitectura do sistema proposto

3.1. Informação a suportar

Como já foi mencionado, o objectivo do sistema proposto é fornecer um dicionário *online* de termos de várias áreas científicas. Para a criação desse dicionário é necessário estruturar a informação que é utilizada na caracterização de um termo.

Essa estrutura é conseguida através da análise de um exemplo de um termo e de todo o conteúdo que o define. Tomando em consideração, numa primeira fase, a seguinte informação referente ao termo *Infectologia*:

> **infectología f.** (Medicina)
>
> Especialidade da medicina interna que estuda a prevenção, diagnóstico e prognóstico das enfermidades que são produzidas por agentes infecciosos.

Analisando esta informação é possível diferenciar alguns atributos de informação que participam na sua caracterização como um género gramatical, ou seja, se um termo é masculino, ou feminino ou neutro; uma área científica, ou seja, a ciência a que o termo pertence. Juntamente com estes atributos, foi requisitado suporte para a classe gramatical de um termo. Resumindo, existem 4 atributos distintos que participam na caracterização de um termo:

Termo:

- Género gramatical
- Classe gramatical
- Ciência
- Definição

Como mencionado anteriormente, a aplicação proposta é um dicionário etimológico e histórico, sendo por isso necessário apresentar suporte para tal informação. A informação referente à etimologia, é obtida pela segmentação e identificação dos constituintes morfológicos, ou seja, dos lexemas, dos sufixos e dos prefixos que o compõem. Esta situação pode ser verificada

na continuação do exemplo anterior, agora com a apresentação dos constituintes morfológicos do termo *Infectologia*:

infectología f. (Medicina)

(...)

Etimologia: ***in-fēc-***[1] lat. 'introduzir', 'misturar' + ***-tu(m)*** [2] lat. + ***logí(ā)*** [3] -λογία gr. 'estudo'

1 e 3 – Lexemas , 2 - sufixo

A informação relativa a um sufixo, a um prefixo e a um lexema, apesar de participarem na caracterização de um termo, não devem ser considerados como atributos, mas sim como um grupo de informação que possui um conjunto de atributos que os caracteriza. Por esta razão, é necessário analisar a informação que cada um dos constituintes morfológicos apresenta. Considerando o sufixo presente no exemplo anterior, a informação relativa a esse sufixo é a seguinte:

2 – Sufixo

-tu(m) lat., entra na formação de substantivos.

Como é possível observar, o sufixo vai apresentar um atributo que caracteriza a sua língua e um atributo que contém a sua definição. É ainda necessário considerar que um sufixo pode apresentar um significado gramatical, que indica a sua classificação segundo a sua distribuição sintáctica e morfológica e um significado semântico referente ao estudo do significado. Estes atributos são também aplicáveis no caso de um prefixo.

Resumidamente, a caracterização de um sufixo e de um prefixo engloba os seguintes atributos:

Prefixo e Sufixo:

- Língua
- Definição
- Significado Semântico
- Significado Gramatical

Para o lexema, a análise é mais complexa:

1 (Lexema)

in-fēc- lat. (verbo), 'introduzir', 'misturar'

Um lexema é formado de um étimo, que por sua vez está ligado a um radical indo-europeu:

Étimo: *in-fēc-* lat

Raiz: **dheə-/*dhə-*, do indoeuropeu, 'colocar', 'fazer'

Para além de apresentar um significado e uma classe gramatical, um lexema é formado por um étimo, que diz respeito às formas gregas e/ou latinas. Este atributo, o étimo do lexema regista as mesmas palavras, mas com as suas formas gregas e/ou latinas. Por esta razão, é necessário separar a informação de um étimo da informação de um lexema, prevenindo os casos em que a definição pode não ser a mesma, tal como a palavra a registar, que se encontra dependente da língua do étimo. Este caso acontece para o outro lexema presente no exemplo do termo *infectologia*:

3 (lexema)

logí(ā) ***-λογία*** gr. (subst.), 'estudo'

O étimo que corresponde a este lexema é o seguinte:

Etimo: ***-λογία*** **gr.** estudo

Raiz: **leg-*, indoeuropeu, 'colher'

Como já foi mencionado, um étimo está ligado a um radical indo-europeu. O radical ou raiz reporta à forma primitiva que está na base dos termos greco-latinos. A partir desta informação e dos exemplos anteriores, é possível descrever os atributos e grupos de informação que caracterizam um lexema:

Lexema:

- Definição ou conceito
- Classe Gramatical
- *Étimo*:
  - Definição
  - Língua
  - *Radical ou raiz*
    - Definição
    - Língua

Finalmente, como o sistema é também um dicionário histórico, é necessário suportar a informação relativa ao histórico do termo. A informação contida no histórico de um termo inclui a língua-base, o respectivo tipo de classificação histórica, ou seja, se um termo é um neologismo ou uma palavra antiga e a data em que ele surgiu pela primeira vez, com referência ao século e ano. Para completar esta informação também convém indicar a origem dessa informação, ou seja, a fonte de onde foi obtida, para futura referência.

Com a análise das características de um termo concluída, é possível exibir uma lista com todos os atributos e grupos de informação que irão participar na caracterização do termo.

Termo:

- Definição
- Classe Gramatical
- Género
- Ciência
- *Prefixo*
    - Língua
    - Definição
    - Significado Semântico
    - Significado Gramatical
- *Lexema*
    - Definição ou conceito
    - **Étimo:**
        - Definição
        - Língua
        - *Radical ou raiz*
            - Definição
            - Língua
- *Sufixo*
    - Língua
    - Definição
    - Significado Semântico
    - Significado Gramatical
- Língua – base;
- Classificação histórica
- Fonte
- Século
- Ano

3.2. Funcionalidades

As funcionalidades requisitadas para a aplicação foram divididas em módulos, agrupados em duas categorias: visualização de informação e inserção de informação. Os módulos pertencentes a categoria de visualização da informação armazenada na base de dados são os seguintes:

Visualização da informação	Objectivo
Ciência	O utilizador visualiza a informação de uma Ciência sendo também possível visualizar os seus subdomínios científicos caso existam.
Prefixo	O utilizador visualiza a informação de um Prefixo.
Sufixo	O utilizador visualiza a informação de um Sufixo.
Lexema	O utilizador visualiza a informação de um Lexema, exibindo o étimo do lexema e o radical indo-europeu que correspondente.

Para os módulos anteriores, ainda é apresentada uma lista de todos os termos que pertencem a um dada ciência e que utilizam um dado prefixo, sufixo ou lexema. Esta listagem é feita em forma de hiperligações para possibilitar a visualização do termo seleccionado.

Para além dos módulos anteriores ainda existe o módulo para visualização de um termo. Este módulo apresenta toda a informação que um dado termo possui, como a Ciência, os constituintes morfológicos e o histórico. Ao contrário do que acontece para os módulos mencionados anteriormente, este módulo irá apresentar hiperligações para a informação referente à Ciência (e subdomínio científico, caso exista) e para os constituintes morfológicos.

Estes módulos também podem ser visualizados através da utilização de um diagrama de casos de uso. Um diagrama de casos de uso é uma representação externa do sistema. Nele são representados graficamente os actores externos ao sistema, neste caso é apresentado o utilizador que irá interagir com os casos de uso. Os casos de uso correspondem às funcionalidades que o sistema apresenta..

O diagrama de casos de uso que corresponde à categoria de visualização é o seguinte:

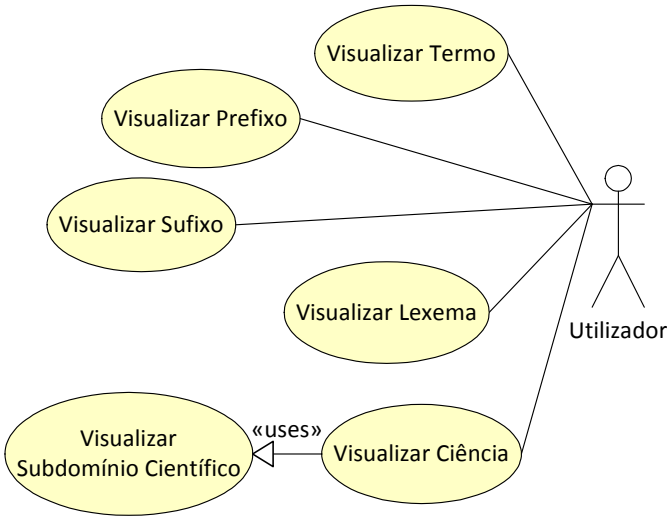


Imagem 1 - Diagrama de casos de uso para a visualização de informação

De realçar no diagrama anterior a relação existente entre o caso de uso *Visualizar Ciência* e o caso de uso *Visualizar Subdomínio Científico*. Esta relação, como já tinha sido indicado, ocorre porque o caso de uso Visualizar Ciência permite a visualização dos subdomínios científicos que estão associados a uma dada ciência.

Para a categoria de inserção de informação na base de dados os módulos definidos são os seguintes:

Inserção da informação	Objectivo
Ciência	O utilizador insere toda a informação relativa a uma ciência.
Subdomínio Científico	O utilizador insere toda a informação relativa a um subdomínio científico pertencente a uma dada ciência. Este módulo permite a inserção de uma nova ciência, caso seja necessário.
Prefixo	O utilizador insere toda a informação relativa a um prefixo.
Sufixo	O utilizador insere toda a informação relativa a um sufixo.
Lexema	Inserir toda a informação relativa a um lexema, com o étimo e o radical correspondente ao étimo.

Uma vez mais, o módulo para inserção de termos apresenta algumas diferenças em relação aos outros módulos de inserção. Para além de permitir a inserção de um novo termo, este módulo permite a utilização de informação já existente na base de dados (como por exemplo para a ciência e para os constituintes morfológicos). Caso essa informação, ou outra que caracteriza um termo não exista, o módulo permite a sua inserção.

O diagrama de casos de uso para a categoria de inserção é o seguinte:

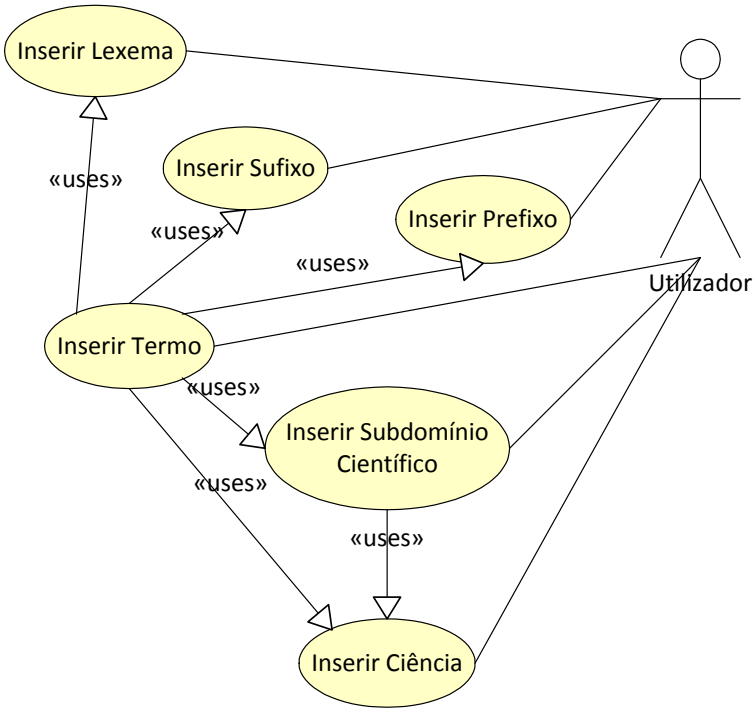


Imagem 2 - Diagrama de casos de uso para a inserção de informação

De realçar que o caso de uso *Inserir Termo* possibilita a inserção de novos dados referentes a restante informação existente na base de dados. De notar ainda a relação entre o caso de uso *Inserir Subdomínio Científico* com o caso de uso *Inserir Ciência*, uma vez que o primeiro caso de uso possibilita a inserção de uma nova ciência para ser associada a um dado subdomínio científico.

Para além desta divisão em categorias, referente às operações de visualização e inserção da informação, ainda existe mais um conjunto de funcionalidades que não se encaixa nas categorias anteriores, funcionalidades essas que são exibidas na página principal da aplicação. Esse conjunto de funcionalidades está relacionado com as várias formas de pesquisa que é possível realizar. Como é lógico, o sistema apresenta uma pesquisa por termos que devolve uma lista com hiperligações dos termos que respeitam os critérios da pesquisa. Esta forma de pesquisa tem a possibilidade de funcionar com *wildcards* (através do caracter asterisco) para alargar a abrangência dos critérios da pesquisa. Para além deste tipo de pesquisa por termos, também existe a possibilidade de os pesquisar pela ciência a que eles pertencem.

Da mesma forma, o sistema possibilita a pesquisa por prefixos e por sufixos. Uma pesquisa deste género pode ser realizada alfabeticamente ou pela selecção do significado semântico e gramatical. Esta pesquisa devolve uma lista com hiperligações de prefixos ou sufixos, dependente do caso, que respeita os critérios escolhidos. A última pesquisa que o sistema suporta é uma pesquisa por lexemas. Esta pesquisa apresenta a possibilidade de pesquisa alfabética e pesquisa livre, através da definição de critérios a pesquisar. Esta última forma, tal como a pesquisa por termos, apresenta a capacidade da utilização de *wildcards*. Mais uma vez, os resultados da pesquisa são exibidos numa lista de hiperligações de lexemas, que possibilitam visualização completa do lexema seleccionado.

O diagrama de casos de uso que expressa as pesquisas mencionadas, é o seguinte:

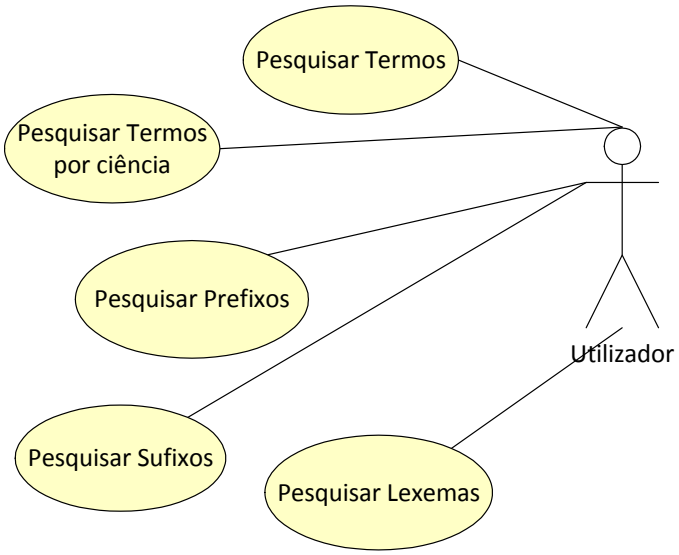


Imagem 3 - Diagrama de casos de uso para as pesquisas

3.3. Base de Dados:

3.3.1. Classes de informação

De forma a construir uma base de dados fiável e segura, a partir da informação definida no subcapítulo 3.1. é necessário fazer a transformação dessa informação de uma forma cuidada e faseada. Numa primeira fase é necessário verificar como é possível agrupar a informação classes distintas. Cada uma destas classes deve ser caracterizada por um conjunto de atributos.

Por isso, a partir da lista exibida no subcapítulo 3.1. é possível realizar uma divisão em classes de forma a criar uma organização mais compreensível. Relembrando então toda a informação que necessita se suportada:

Termo:

- Definição
- Classe Gramatical
- Género
- Ciência
- *Prefixo*
    - Língua
    - Definição
    - Significado Semântico
    - Significado Gramatical
- *Lexema*
    - Definição ou conceito
    - **Étimo:**
        - Definição
        - Língua
        - *Radical ou raiz*
            - Definição
            - Língua
- *Sufixo*
    - Língua
    - Definição
    - Significado Semântico

- Significado Gramatical
- Língua – base;
- Classificação histórica
- Fonte
- Século
- Ano

É fácil perceber que a informação relativa a um termo é representada por uma classe de informação. O mesmo acontece para o prefixo, para o lexema e para o sufixo que vão ser representados por classes distintas e separadas da classe termo. Esta divisão ocorre, porque tanto o prefixo, como o lexema e o sufixo apresentam atributos particulares e podem ser utilizados para a caracterização de termos distintos. Para além destas divisões lógicas assinaladas anteriormente, é preciso ter em atenção que alguns dos atributos que caracterizam um termo apresentam características particulares. Esses atributos precisam de ser separados em classes próprias, uma vez que para além de possuírem características particulares, também podem ser utilizados na caracterização de termos distintos. Esta divisão ocorre para os atributos Classe Gramatical, Ciência e Língua.

Todas estas classes, e o conjunto dos seus atributos, vão ser utilizados para caracterizar um termo, mas para que tal seja possível, é necessário definir que tipos de relações existem entre eles. Nas tabelas seguintes, é possível verificar todas as classes em que a informação foi dividida bem como os atributos que caracterizam cada uma. Também são indicadas as relações que elas apresentam umas com as outras.

Classe para a Classe Gramatical:

A informação relativa a uma classe gramatical é caracterizada pelos seguintes atributos:

Atributo	Descrição
Classe gramatical	É a classificação de uma palavra segundo a sua distribuição sintáctica e morfológica.
Abreviatura	Irá conter a abreviatura da classe.

Tabela 1 - Atributos que caracterizam uma Classe Gramatical

Esta classe só está relacionada com a classe Termo:

Relacionado com:	Relação
Termo	Um termo só pode ter uma classe gramatical.

Tabela 2 - Relações apresentadas por uma Classe Gramatical

Classe Ciência:

A informação relativa a uma Ciência é caracterizada pelos seguintes atributos:

Atributo	Descrição
Ciência	Indica o nome da ciência.
Descrição:	Contem uma descrição da ciência.

Tabela 3 - Atributos que caracterizam uma Ciência

Só existe uma relação com a classe Termo.

Relacionado com:	Relação
Termo	Um termo só pode ter uma ciência.

Tabela 4 - Relações apresentadas por uma Ciência

De acordo com os requisitos, é necessário existir uma separação na área científica em subdomínios, que pertencem a um dada ciência. Esses subdomínios serão colocados numa classe própria, porque uma ciência e um termo podem apresentar vários subdomínios científicos. Os atributos de um subdomínio científico são os seguintes:

Atributo	Descrição
Subcategoria:	Indica o nome da subcategoria científica.
Descrição:	Contém a descrição da subcategoria científica.

Tabela 5 - Atributos que caracterizam um Subdomínio Científico

De acordo com o que foi mencionado anteriormente, os subdomínios estão relacionados com as classes Ciência e Termo.

Relacionado com:	Relação
Ciência	Uma subcategoria só pode pertencer a uma ciência, e uma ciência pode ter várias subcategorias.
Termo	Um termo pode ter várias subcategorias científicas.

Tabela 6 - Relações apresentadas por um Subdomínio Científico

Classe Prefixo e Classe Sufixo:

Os atributos que caracterizam um prefixo, numa análise inicial, são os seguintes:

Atributo	Descrição
Prefixo	O nome do prefixo.
Definição	A definição do prefixo.
Língua	A língua em que surge o prefixo.
Significado Semântico	Relacionado com o sentido do prefixo.
Significado Gramatical	Relacionado com a classe gramatical a que o prefixo pertence.

Tabela 7 - Atributos que caracterizam um Prefixo

E o relacionamento obtido inicialmente é o seguinte:

Relacionado com:	Relação
Termo	Um termo pode ter vários prefixos.

Tabela 8 - Relações apresentadas por um Prefixo

Para a classe Sufixo, o raciocínio é idêntico:

Atributo	Descrição
Sufixo	O nome do sufixo.
Definição	A definição do sufixo.
Língua	A língua em que está o sufixo.
Significado Semântico	Relacionado com o sentido do sufixo.
Significado Gramatical	Relacionado com a classe gramatical a que o sufixo pertence.

Tabela 9 - Atributos que caracterizam um Sufixo

O relacionamento inicial:

Relacionado com:	Relação
Termo	Um termo pode ter vários sufixos.

Tabela 10 - Relações apresentadas por um Sufixo

Depois de uma segunda análise, foi verificado que os significados semânticos e gramaticais podem ser utilizados para caracterizar termos distintos. Por essa razão, essa informação deve ser dividida e colocada em classes separadas:

Classes Significado Semântico e Gramatical para um prefixo

Atributo	Descrição
Significado Semântico	Relacionado com o sentido do prefixo.

Tabela 11 - Atributos que caracterizam o Sign. Semântico de um Prefixo

Atributo	Descrição
Significado Gramatical	Relacionado com a classe gramatical a que o prefixo pertence.

Tabela 12 - Atributos que caracterizam o Sign. Gramatical de um Prefixo

O relacionamento destas duas últimas classes é lógico:

Relacionado com:	Relação
Prefixo	Um prefixo só pode ter um Significado Semântico e um Significado Gramatical.

Tabela 13 - Relações apresentadas para os Sign. Semânticos e Gramaticais de um Prefixo

Classes Significado Semântico e Gramatical para um sufixo

Para o caso de um sufixo, o raciocínio é idêntico ao utilizado para um prefixo:

Atributo	Descrição
Significado Semântico	Relacionado com o sentido do sufixo.

Tabela 14 - Atributos que caracterizam o Sign. Semântico de um Sufixo

Atributo	Descrição
Significado Gramatical	Relacionado com a classe gramatical a que o sufixo pertence.

Tabela 15 - Atributos que caracterizam o Sign. Gramatical de um Sufixo

Os seus relacionamentos são os seguintes:

Relacionado com:	Relação
Sufixo	Um sufixo só pode ter um Significado Semântico e um Significado Gramatical.

Tabela 16 - Relações apresentadas para os Sign. Semânticos e Gramaticais de um Sufixo

Classes Lexema, Étimo e Raiz do étimo:

Tendo em conta que um lexema é formado por um étimo, que por sua vez se liga a um radical indo-europeu surgem três classes distintas, mas relacionadas entre si.

Os atributos que caracterizam um Lexema são os seguintes:

Atributo	Descrição
Lexema	O nome do lexema.
Conceito	O conceito do lexema.

Tabela 17 - Atributos que caracterizam um Lexema

Um lexema para além de estar relacionado com a classe étimo (este relacionamento foi definido na classe étimo), também se encontra relacionado com a classe Termo.

Relacionado com:	Relação
Termo	Um termo pode ter vários lexemas.

Tabela 18 - Relações apresentadas por um Lexema

Os atributos para a classe Étimo são os seguintes:

Atributo	Descrição
Étimo	Diz respeito às formas gregas e/ou latinas. Neste campo são registadas as mesmas palavras que no lexema, mas com a sua forma grega e latina.
Definição	Definição do étimo.
Língua	Língua do étimo.

Tabela 19 - Atributos que caracterizam um Étimo

Aqui está a relação com a classe Lexema.

Relacionado com:	Relação
Lexema	Um lexema só pode ter um étimo.

Tabela 20 - Relações apresentadas por um Étimo

Os atributos que caracterizam a classe Raiz são os seguintes:

Atributos	Descrição
Raiz	Diz respeito à forma primitiva (indo-europeia) que está na base dos termos greco-latinos.
Definição	Definição da raiz.
Língua	Língua da Raiz.

Tabela 21 - Atributos que caracterizam um Radical indo-europeu

A sua relação com a classe Étimo:

Relacionado com:	Relação
Étimo	Um étimo só pode ter uma raiz.

Tabela 22 - Relações apresentados por um Radical indo-europeu

Classe Termo:

Por fim, os atributos que caracterizam a Classe Termo:

Atributo	Descrição
Termo	O nome do termo.
Definição	A definição do termo.
Género	O género do termo.
Língua Base	Indica qual é a língua base do termo.
Classificação	Indica o tipo do termo no aspecto cronológico.
Fonte	Contém a origem da informação apresentada pelo termo.
Século	O século em que o termo surgiu.
Ano	O ano em que o termo surgiu.

Tabela 23 - Atributos que caracterizam um Termo

Não é necessário indicar nenhuma relação uma vez que já foram definidas nas classes anteriores.

Analisando os atributos descritos anteriormente, é possível verificar que o atributo Língua Base, poderá pertencer a vários termos diferentes. De notar também que nas Classes Prefixo, Sufixo, Étimo e Raiz, existe um atributo Língua. Devido a estes dois factos, foi criada uma classe separada para a Língua, sendo o atributo língua existente retirado das classes indicadas.

Atributo	Descrição
Língua	Língua.
Abreviatura	Abreviatura da língua.

Tabela 24 - Atributos que caracterizam uma Língua

As relações da classe Língua com as restantes Classes são simples de compreender, pois cada um dos termos, dos prefixos, dos sufixos, dos étimos e dos radicais só poderá apresentar uma língua.

3.3.2. Diagrama de Classes

A partir das classes definidas anteriormente e das relações existentes entre elas, é fácil construir um diagrama de classes. Este diagrama de classes vai permitir a construção do modelo da base de dados de uma forma mais rápida e eficiente.

Com as classes definidas para criar o diagrama de classes, só é necessário analisar as relações existentes para descobrir como estas vão ser apresentadas no diagrama. Por exemplo, a relação existente entre a classe Termo e Língua no diagrama de classes vai ser transformada numa associação binária de muitos (do lado da classe Termo) para 1 (do lado da classe Língua). O diagrama de classes para estas duas classes é o seguinte:

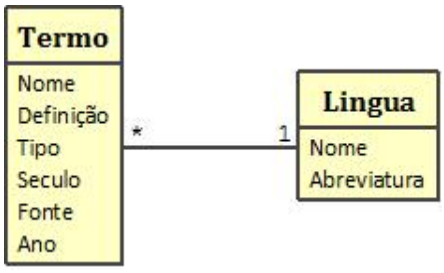


Imagem 4 - Diagrama de Classes: Classes Termo e Língua

Existem mais relações deste tipo. A transformação dessas relações para o diagrama é igual à apresentada. Essas relações são as seguintes:

Termo	<-->	Classe Gramatical
Termo	<-->	Ciência
Lexema	<-->	Étimo
Étimo	<-->	Raiz
Étimo	<-->	Língua
Sufixo	<-->	Língua
Sufixo	<-->	Sign. Semântico
Sufixo	<-->	Sign. Gramatical
Prefixo	<-->	Língua
Prefixo	<-->	Sign. Semântico
Prefixo	<-->	Sign. Gramatical

Tabela 25 - Diagrama de Classes: Relações de muitos para um

Uma relação diferente ocorre, por exemplo, entre a classe Termo e classe Lexema. Quando numa relação, um termo pode ter vários lexemas, surge no diagrama de classes uma associação de muitos para muitos:

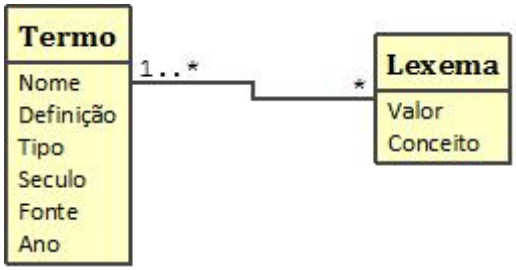


Imagem 5 - Diagrama de Classes: Classes Termo e Lexema

Existem mais relações deste género, que ocorrem entre as classes:

Termo	<-->	Prefixo
Termo	<-->	Sufixo
Termo	<-->	Subcategoria
Ciência	<-->	Subcategoria

Tabela 26 - Diagrama de Classes: Relações de muitos para muitos

Como não existe outro tipo de relações, a construção do diagrama de classes final é obtido facilmente.

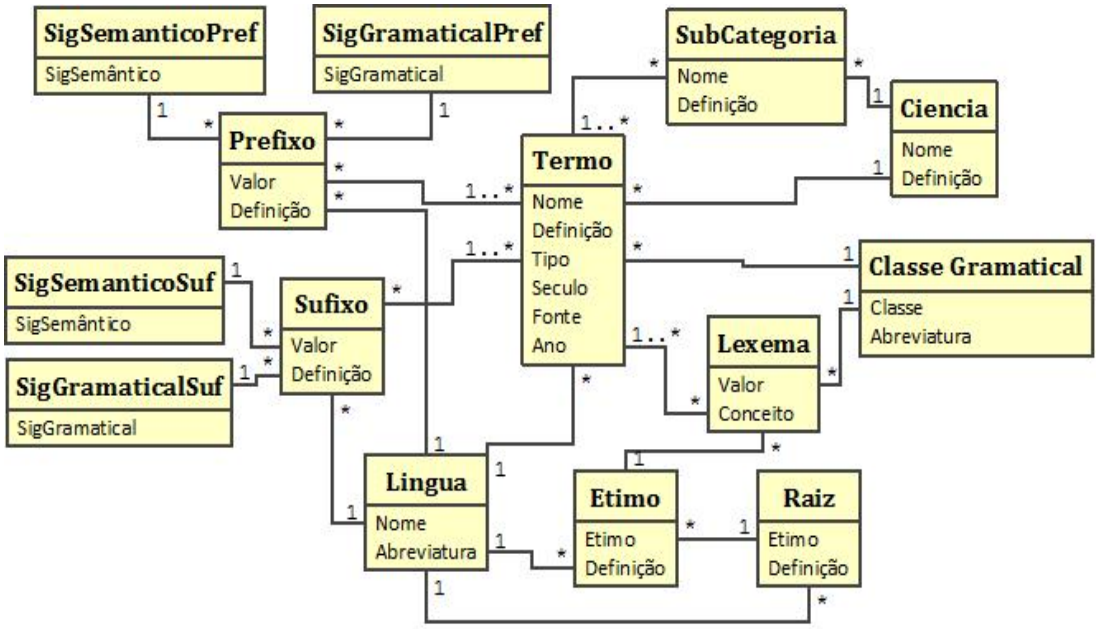


Imagem 6 - Diagrama de Classes completo

3.3.3. Modelo físico

A partir do diagrama de classes é possível construir um modelo físico da base de dados, que tornará mais fácil, a construção das respectivas tabelas.

A conversão do diagrama de classes, para o modelo físico de uma base de dados, apresenta um conjunto de regras relacionadas com a conversão das relações de um modelo para o outro. É atribuída a cada tabela uma chave primária que é o identificador único dos dados presentes na tabela (o *id*), podendo ser uma coluna ou um conjunto de colunas; e caso seja necessário, chaves estrangeiras que representam as relações com outras tabelas, referenciando a chave primária da outra tabela que participa na relação.

No diagrama de classes existem apenas dois tipos de relações entre as classes: associação de muitos para um e de muitos para muitos. Quando a associação é do género de muitos para um (*:1), são necessárias duas tabelas: uma por cada classe. As tabelas daí resultantes terão a sua chave primária e a tabela do lado dos muitos terá como chave estrangeira, a chave primária da outra tabela.

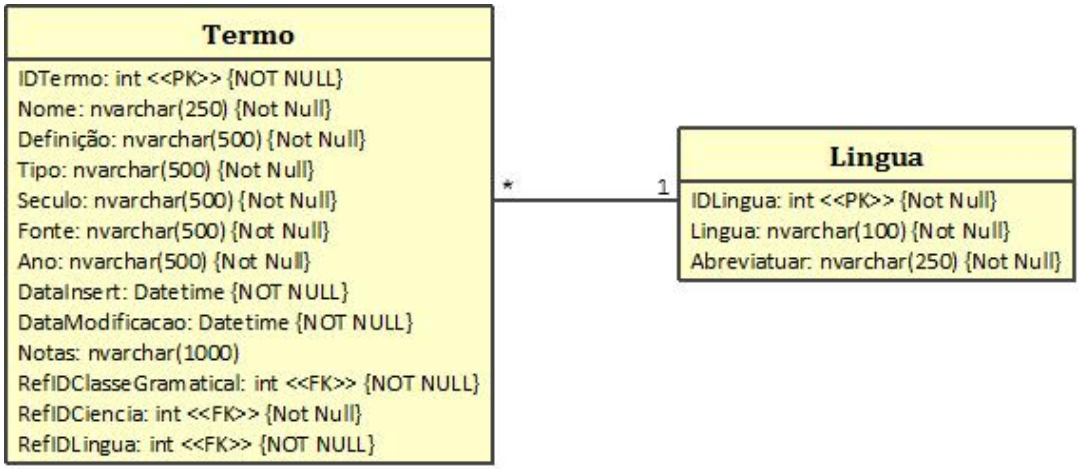


Imagem 7 - Modelo físico: Termo (inicial) e Língua

Como é possível verificar na figura apresentada acima, é no modelo físico que é indicado o tipo de cada atributo. Da mesma foram, é indicado qual dos atributos é a chave primária, através da utilização da sigla PK (*Primary Key*) e quais serão as chaves estrangeiras, que representam as relações existentes, através da utilização da sigla FK (*Foreign Key*). O tipo dos dados é exibido à frente de cada atributo indicando (quando aplicável) o tamanho máximo que eles irão apresentar.

Quando a associação é de muitos para muitos (1..*:*) são necessárias três relações: uma por cada classe, resultando em tabelas com as respectivas chaves primárias. A terceira tabela vai corresponder à associação. Esta tabela vai apresentar duas chaves estrangeiras, uma por cada chave primária das tabelas participantes na associação. A respectiva chave primária vai ser composta por estas duas chaves secundárias.

O diagrama seguinte demonstra a relação acabada de descrever:

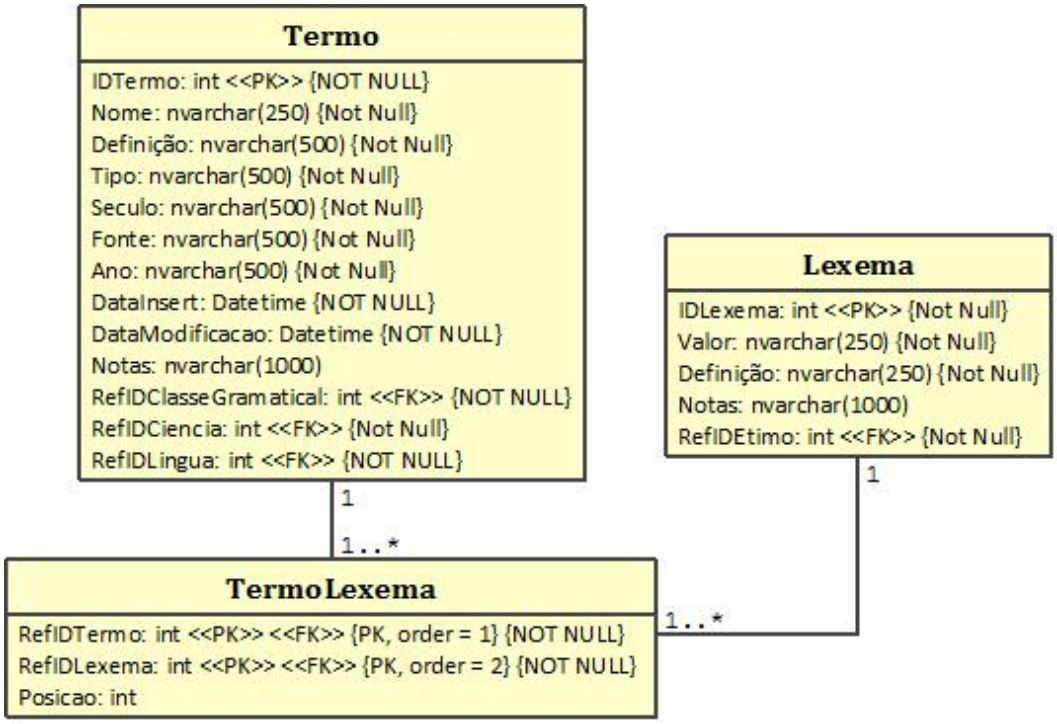


Imagem 8 - Modelo Físico: Termo, TermoLexema e Lexema

Na tabela TermoLexema (ver Imagem 8), aparece um novo campo denominado *Posicao*. Este campo vai ser utilizado para armazenar a posição relativa dos lexemas. Por exemplo, caso existam dois lexemas associados a um termo, este campo permite armazenar a ordem que eles apresentam no termo. Este campo vai existir em todas as tabelas que relacionam um termo com o seu constituinte morfológico e vai ser útil na construção da ordem que estes apresentam num dado termo.

Com o decorrer da construção da aplicação foi necessário realizar algumas alterações na base de dados, alterando o modelo físico. Para uma melhor compreensão e antes de analisar a criação das tabelas, é necessário indicar que alterações foram feitas e as respectivas repercussões no modelo físico da base de dados.

Chegou-se à conclusão de que a informação relativa à cronologia do termo, ou seja, o tipo, a fonte, a data e o século poderá não estar disponível para um dado termo. Por esta razão, essa informação foi separada da tabela Termo e colocada numa tabela própria. De notar que esta tabela permite a inserção de nulos para todos os campos, visto que alguns dos campos poderão não ter informação disponível.

A nova tabela Termo com estes atributos retirados e a nova tabela, denominada Cronologia, são as seguintes:

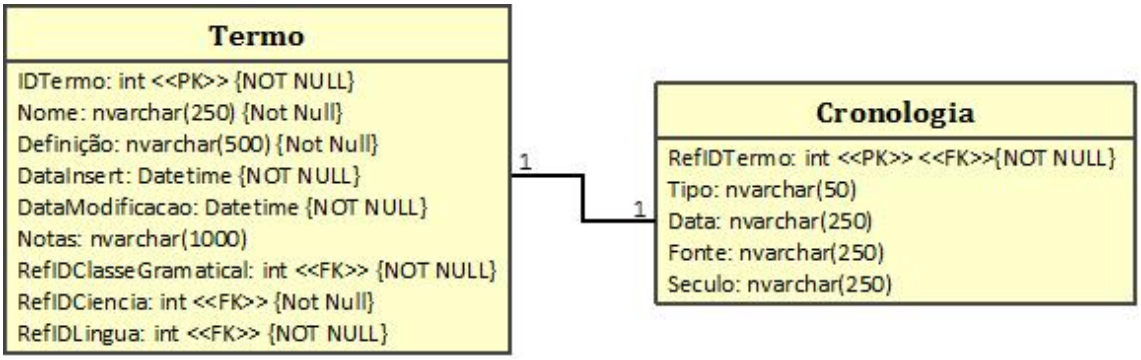


Imagem 9 - Modelo Físico: Termo e Cronologia

Foram adicionadas mais duas tabelas, que não foram definidas no diagrama de classes e que estão relacionadas com as tabelas Prefixo e Sufixo, para armazenar o nome em grego, que cada um deles pode apresentar, caso pertençam à língua Grega.

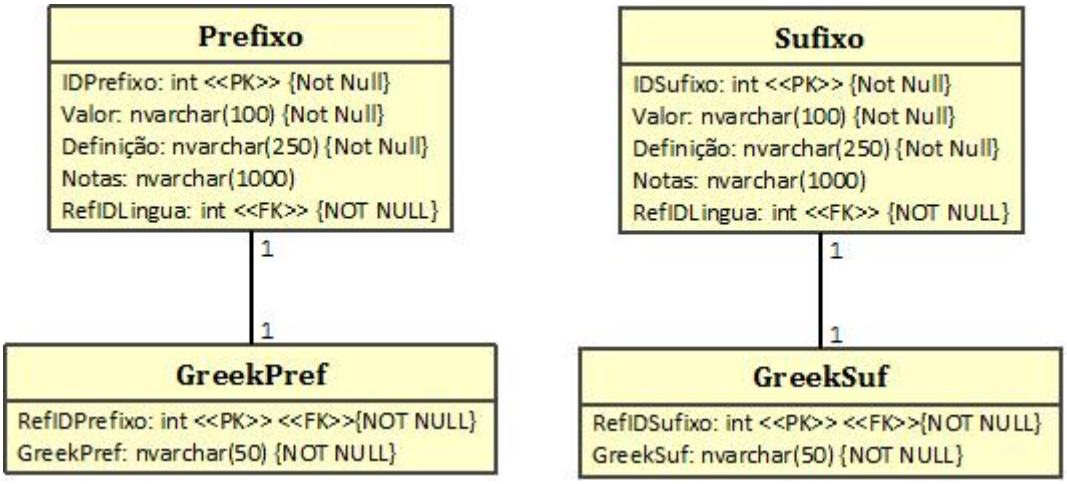


Imagem 10 - Modelo Físico: Prefixo e Sufixo e respectivas tabelas para o Grego

Para a visualização da informação dos constituintes morfológicos, foi criada mais uma tabela que irá armazenar a ordem global de todos os constituintes. É através desta e das tabelas específicas para cada constituinte, mencionadas anteriormente que a ordem dos constituintes é armazenada.

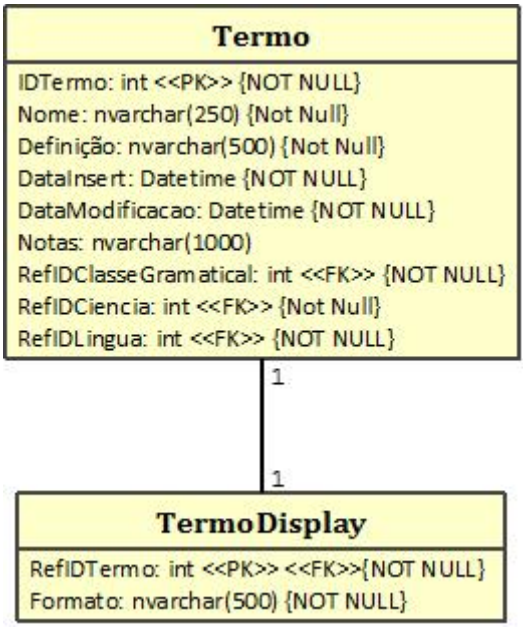


Imagem 11 - Modelo Físico: Termo e TermoDisplay

Finalmente, foram adicionadas duas tabelas com o objectivo de armazenar o número de vezes que um termo é pesquisado por mês.

Depois destas alterações, o modelo físico final pode ser visualizado no anexo A. E alguns exemplos do código de implementação das tabelas e relações apresentadas pelo modelo físico é incluindo no anexo B.

3.4. Arquitectura

O sistema proposto segue uma arquitectura cliente - servidor simples. Como o sistema é centrado na Web, a comunicação entre as entidades envolvidas é realizada através da Internet. O modelo que demonstra esta arquitectura é bastante simples como é possível verificar na imagem seguinte.

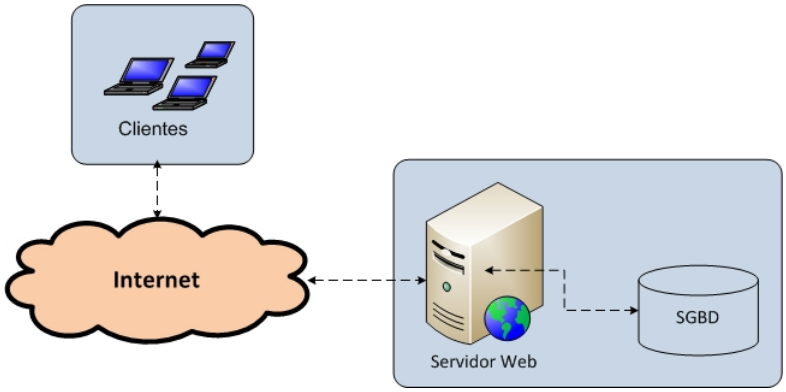


Imagem 12 - Modelo Cliente - Servidor

Servidor:

Neste sistema, o lado do servidor é composto por dois componentes: um servidor Web e um Sistema de Gestão de Bases de Dados (SGBD). Não é obrigatório que estes dois componentes estejam na mesma máquina nem que se encontrem no mesmo local, mas para aumentar o desempenho da aplicação é aconselhável que o Sistema de Gestão de Base de Dados se encontre no mesmo local físico que o servidor Web.

É no servidor Web que a aplicação vai estar armazenada. Este recurso vai receber e processar os pedidos provenientes do cliente. Como a aplicação Web foi desenvolvida com a tecnologia ASP .NET, o servidor envia código HTML como respostas aos pedidos do cliente. Este código HTML é gerado pelo conjunto do código ASP .NET e pelos dados existentes no pedido. Este servidor também é responsável por executar pedidos de informação ao sistema de gestão de base de dados sendo a resposta a esses pedidos de informação, utilizada na construção do código HTML respectivo a enviar para o cliente.

O sistema de gestão de base de dados é então responsável pelo armazenamento da informação e pela resposta aos pedidos de informação provenientes do servidor Web.

Cliente:

Como a aplicação foi desenvolvida na tecnologia ASP .NET, o cliente é um *browser* de Internet. O cliente para visualizar uma página requisita-a ao servidor, enviando um pedido com todos os dados necessários encapsulados. O resultado desse pedido ao servidor, como já foi mencionado, surge sob a forma de código HTML, que irá gerar a página requisitada, no *browser*.

Os papéis despenhados pelo cliente e pelo servidor acima descritos, são característicos do modelo de uma aplicação Web clássica, como apresentado na imagem seguinte:

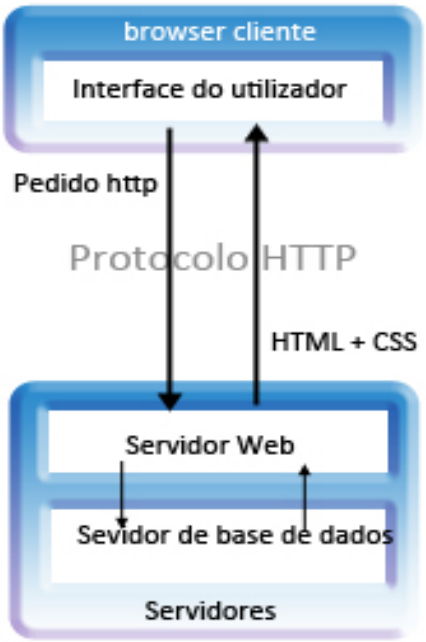


Imagem 13 – Arquitectura: aplicação Web Clássica

Mas devido à utilização de AJAX em alguns dos módulos da aplicação, a comunicação entre cliente e servidor é efectuada de um modo diferente, como pode ser observado na imagem seguinte:

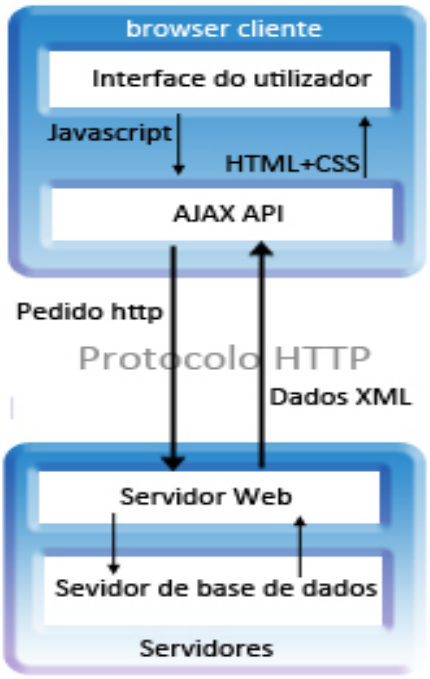


Imagem 14 - Arquitectura: aplicação Web com AJAX

Este modelo apresenta algumas diferenças em relação ao modelo Web clássico. De uma forma simplista, o *browser* do cliente, neste modelo, interage com o AJAX API para obter a interface desejada. Essa interacção é feita através da utilização de javascript para os pedidos. Por sua vez, a AJAX API realiza os pedidos para o servidor Web através de pedidos http. Estes são respondidos através da utilização de XML, processados pela AJAX API e enviados para o *browser* do cliente, uma vez mais, em código HTML, possibilitando a construção do interface por parte do *browser*. Graças a esta forma de funcionamento, é possível realizar pedidos de informação ao servidor sem ser necessário esperar pela resposta, ou no caso de aplicações Web, sem ser necessário um refrescamento completo da página (funcionamento assíncrono).

3.5. Construção da aplicação Web

O DICITER foi desenvolvido, utilizando tecnologias fornecidas pela Framework .NET 3.5 e mais especificamente o ASP .NET, através da utilização do C# como linguagem de programação. Como *software* de desenvolvimento de aplicações foi utilizado o Microsoft Visual Studio 2008. Foram ainda utilizadas tecnologias como Javascript e ASP. Net AJAX, de forma a melhorar a usabilidade e aspecto da mesma.

Como já mencionado, a construção da aplicação foi realizada em módulos, agrupados essencialmente em duas categorias, os módulos de visualização de informação e os módulos de inserção de informação, separados fisicamente em pastas distintas: a pasta *selects* para a visualização e *inserts* para a inserção. Para manter a coerência no aspecto ao longo de toda a aplicação, foi utilizada a mesma estrutura e forma de funcionamento. Aspectos como a visualização, inserção dos dados e validação dos mesmos seguem sempre a mesma implementação e construção.

A cada um dos módulos correspondem dois ficheiros: um ficheiro, com extensão aspx, onde é definido o aspecto gráfico ou interface, e segundo o modelo *code-behind*, um ficheiro com extensão aspx.cs, que apresenta o mesmo nome que o ficheiro anterior, e onde é definida toda a lógica de funcionamento que o módulo apresenta. O nome do ficheiro de interface correspondente a cada um dos módulos pertencentes a cada categoria é indicado nas tabelas seguintes:

Visualização de informação:

Ficheiro de interface	Módulo
ViewPrefixo.aspx	Visualizar Prefixo
ViewSufixo.aspx	Visualizar Sufixo
ViewLexema.aspx	Visualizar Lexema
ViewTermo.aspx	Visualizar Termo

Inserção de informação:

Ficheiro de interface	Módulo
InsertCiencia.aspx	Inserir Ciência
InsertSubCat.aspx	Inserir Subdomínio Cientifico
InsertPrefixo.aspx	Inserir Prefixo
InsertSufixo.aspx	Inserir Sufixo
InsertLexema.aspx	Inserir Lexema
InsertTermo.aspx	Inserir Termo

A estrutura apresentada em cada módulo é definida através da utilização do conceito de *Master Page*. O objectivo da utilização de uma *Master Page* é a definição de uma estrutura seguida por todos os módulos, proporcionando uma melhor organização dos conteúdos e dotando a aplicação de um aspecto uniforme. A estrutura definida é a seguinte:

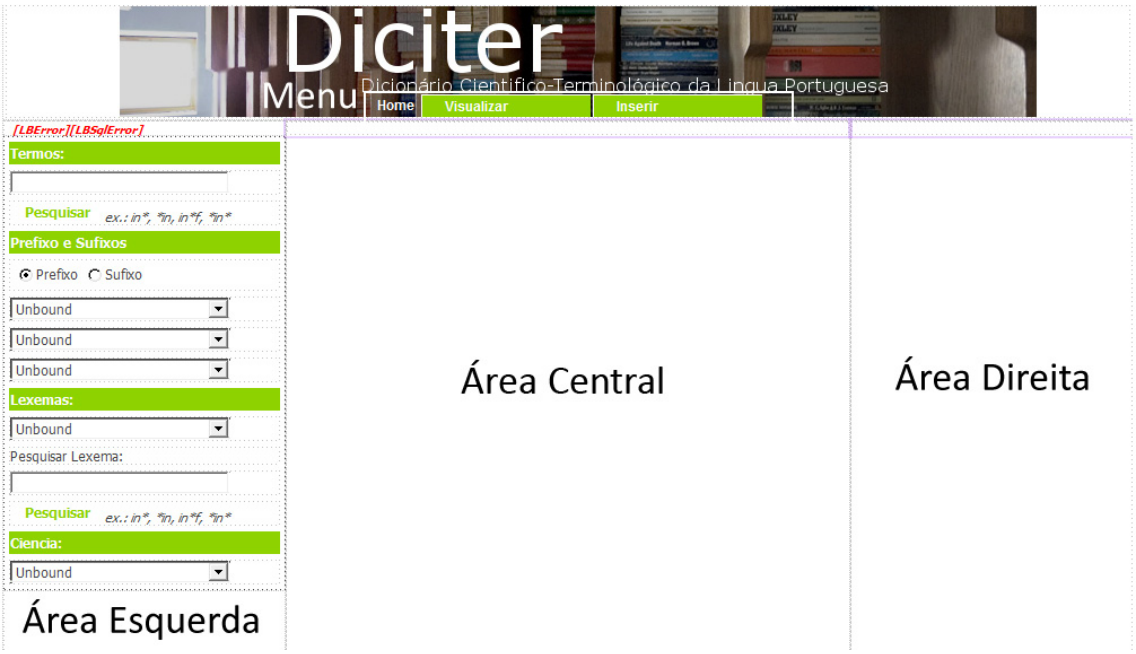


Imagem 15 - Aplicação Web: Áreas definidas

Foram definidas três áreas distintas: uma área esquerda, uma área central e uma área direita. Para além destas áreas, a imagem superior bem como o menu foram também definidos neste ficheiro.

Área Esquerda:

A área esquerda vai apresentar controlos, que são definidos na própria *Master Page* e que estão relacionados com a pesquisa de informação. Esses controlos possibilitam a inserção ou escolha da informação que se deseja pesquisar. Por exemplo, para pesquisar um termo, basta preencher a caixa de texto correspondente e clicar no botão pesquisar.

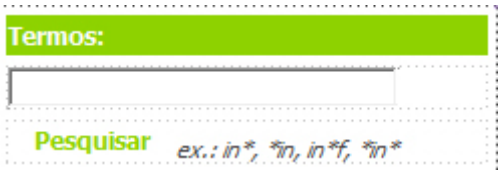


Imagem 16 - Aplicação Web: Pesquisa por termos

Esta pesquisa pode apresentar um *wildcard*, através do caracter asterisco de forma a aumentar a abrangência da pesquisa. Este mesmo funcionamento é utilizado para uma pesquisa por lexemas:

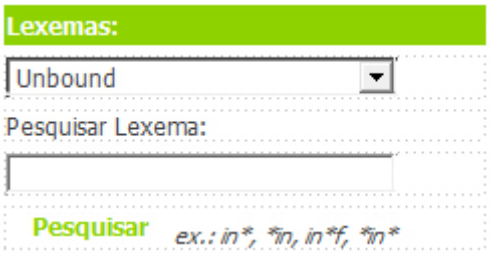


Imagem 17 - Aplicação Web: Pesquisa por Lexemas

Esta área apresenta mais controlos que permitem pesquisar por outro tipo de informação como Sufixos, Prefixos e Lexemas. Para este tipo de informação são apresentados controlos do género de *asp:DropDownList*, inicializados com a informação correspondente na leitura da página. Para efectuar a pesquisa neste tipo de controlos basta seleccionar a informação desejada. Estes controlos não necessitam de nenhum botão para submeter os dados para o servidor a fim de serem processados, sendo isso realizado automaticamente, uma vez que os controlos após sofrerem uma alteração do item seleccionado, realizam o mecanismo de *autopostback*, ou seja, os dados são automaticamente submetidos para o servidor para processamento. A aplicação irá então reagir de acordo com os resultados obtidos.

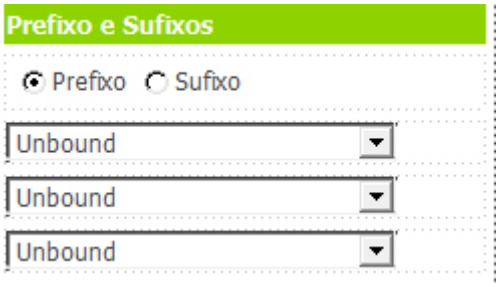


Imagem 18 - Aplicação Web: Pesquisa por Prefixos ou Sufixos

Como exemplo da forma como todos estes controlos são definidos, no Anexo C1, é apresentada a definição dos controlos exibidos na imagem acima.

Finalmente para a pesquisa de termos através da ciência, a implementação é a mesma:

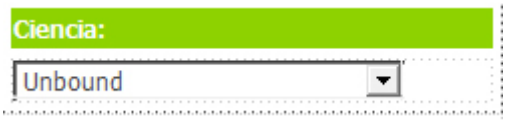


Imagem 19 - Aplicação Web: Pesquisa de termos através da ciência

As operações de pesquisa que cada controlo define não se encontram implementadas na *Master Page*, mas sim na página padrão do *site* – default.aspx. Os controlos apresentados pela *Master Page*, que realizam os pedidos de pesquisa, vão passar para a página default.aspx, através da utilização do conceito de *querystring* do *url*, os parâmetros como o tipo e os dados que devem ser pesquisados. A *querystring* é a parte do *url* que possibilita a passagem de parâmetros entre páginas. A página default.aspx faz o processamento desses parâmetros presentes e realiza a pesquisa que é indicada. Este processamento irá ser analisado mais à frente.

Área central e direita:

As restantes áreas, a área central e a área direita, estão disponíveis para os conteúdos criados pelos módulos de inserção e de visualização. A definição dessas áreas é realizada da mesma forma. Por exemplo, a definição da área central é realizada através da utilização do controlo *asp:ContentPlaceHolder.* Este controlo define uma região para conteúdos, que processa todo o texto, código HTML e controlos que são fornecidos por uma outra página.

Menu:

A *Master Page* define um menu, que possibilita a navegação ao longo do *site*. Este menu é definido através de dois controlos do ASP .NET: o *asp:SiteMapDataSource e o asp:Menu*.

O *asp:SiteMapDataSource* é uma fonte de dados para o mapa do *site*, descrito num ficheiro XML cuja localização é definida na configuração global da aplicação (Web.config). De realçar que é possível a definição de mapas distintos do site em ficheiros XML distintos. Nesta aplicação apenas foi definido um ficheiro com o mapa do site, e é a partir desse ficheiro que o menu é preenchido e torna possível a navegação através dos vários módulos.

Ainda relacionado com o menu, convém mencionar a forma como o seu aspecto gráfico é construído. Esse aspecto é definido através de estilos CSS que são atribuídos aos vários aspectos visuais apresentados pelo menu. Aspectos como a cor, o texto, a alteração de aspecto quando sobreposto pelo rato ou quando se encontra seleccionado, são feitos através da utilização de um ficheiro de *skin* denominado Skins.skin e que se encontra na pasta App_Themes/Skin/. Todo o código de definição do mapa do site, do menu, dos estilos CSS e do ficheiro de *skin* mencionados anteriormente é apresentado no Anexo C2.

Pagina Inicial:

Como já foi mencionado, esta estrutura definida na *Master Page* é utilizada pelos outros módulos, definindo assim um aspecto uniforme para aplicação. O mesmo vai acontecer para a página inicial. A página inicial é construída pelo conjunto da *Master Page* juntamente com a página definida no ficheiro default.aspx. O conteúdo desta página surgirá na área central.

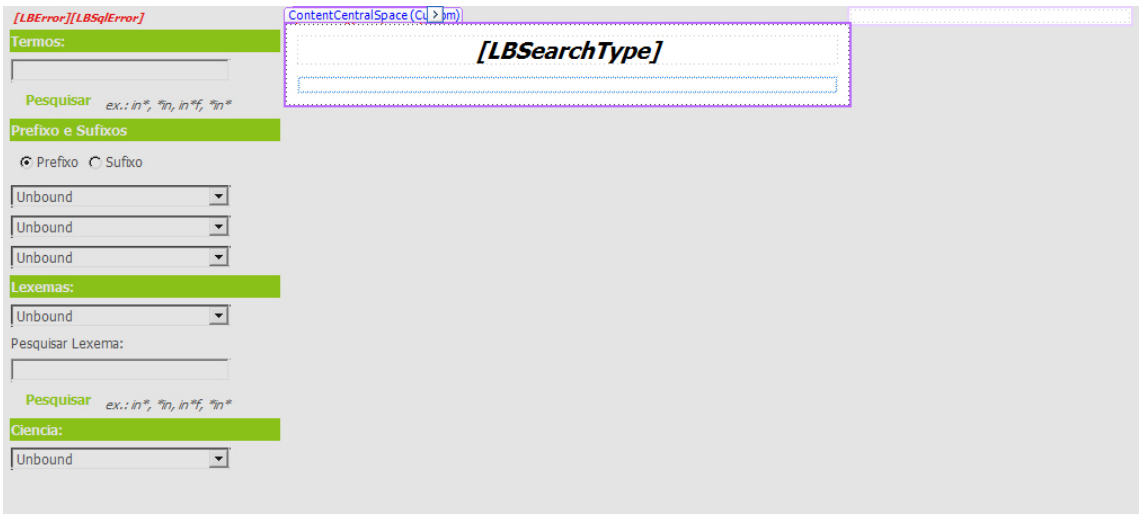


Imagem 20 - Aplicação Web: Página Default.aspx

Na imagem apresentada acima, é possível verificar que os conteúdos definidos na página default.aspx vão ser disponibilizados na área central definida na *Master Page*. É de notar que os conteúdos que são definidos na área esquerda pela *Master Page* estão disponíveis. A colocação

dos conteúdos definidos na página Default.aspx é possível através da utilização do controlo *asp:Content*, colocando no atributo ContentPlaceHolderID, o identificador do controlo asp:ContentPlaceHolder respectivo, definido na *Master Page*.

É dentro deste controlo que são colocados os conteúdos que a página default.aspx define, bem como os que são definidos para os módulos de visualização e de inserção. Nesta página só são definidos dois tipos de controlos: um *asp:Label* que irá apresentar o tipo de pesquisa que foi efectuado e um *asp:Panel*, que é um contentor para outros controlos. É utilizado quando é necessário criar controlos durante o tempo de execução, ou seja, dinamicamente.

A página inicial apresenta duas formas de funcionamento. Uma ocorre quando a aplicação Web é iniciada pela primeira vez, apresentando na área central o último termo que foi inserido na base de dados. A obtenção do último termo inserido na base de dados é realizada através de uma simples *query* de *select* à base de dados, por uma função que pode ser observada no Anexo D1.

A segunda forma de funcionamento ocorre quando é invocada a partir dos controlos de pesquisa existentes na área esquerda, que a invocam utilizando parâmetros na *querystring*. A página default.apsx analisa esses parâmetros de forma a verificar que tipo de pesquisa deve ser realizada. Esta análise é realizada através de uma função própria para o efeito, invocada logo na leitura da página, e como mencionado anteriormente, se existirem parâmetros na *querystring*. A função responsável por este mecanismo é denominada *ParseQueryString* e pode ser consultada no Anexo D2.

A descrição dos controlos existentes na área do lado esquerdo permitem várias formas de pesquisa que convém relembrar:

- Pesquisa por Termos:
    - Por nome;
    - Por Ciência;
- Pesquisa por Prefixos ou Sufixos:
    - Por letra inicial;
    - Por significado semântico ou gramatical;
- Pesquisas por Lexemas:
    - Por letra inicial;
    - Por nome;

De acordo com os resultados da análise da *querystring*, a página Default.apsx, irá proceder à recolha na base de dados dos dados correspondentes. Essa recolha é feita através de simples *queries* de *select* que são realizadas nas tabelas correspondentes. A disponibilização dos

resultados obtidos pode ser feita de duas formas diferentes. Caso exista apenas um resultado que respeite o critério da pesquisa, a página default.aspx redirecciona a aplicação para o módulo de visualização de termos – ViewTermo.aspx. Uma vez mais, a *querystring* é utilizada para enviar o *id* do termo que vai ser visualizado. Mas se a pesquisa devolver um conjunto de termos, estes são colocados dinamicamente no controlo *asp:Panel,* definido anteriormente, sob a forma de controlos do tipo hiperligação, que permitem redireccionar a aplicação Web para o módulo de visualização e ver toda a informação associada a esse termo. A implementação das funções responsáveis pela recolha dos dados respectivos para cada pesquisa é análoga. No anexo D3, é apresentado um exemplo deste mecanismo de recolha de dados, a função para a pesquisa dos dados de um termo. Juntamente com essa função também é indicado a forma como é realizado o preenchimento dinâmico do *asp:Panel* com as hiperligações respectivas.

Arquitectura funcional:

Para compreender melhor o funcionamento das pesquisas mencionadas, é apresentada uma imagem que resume todas a interacções entre o cliente e o servidor Web, e entre este e o SGBD. Considerando que na imagem, a página inicial do site já se encontra totalmente aberta e funcional no *browser* do cliente, o funcionamento de qualquer uma das pesquisas possíveis é o seguinte:

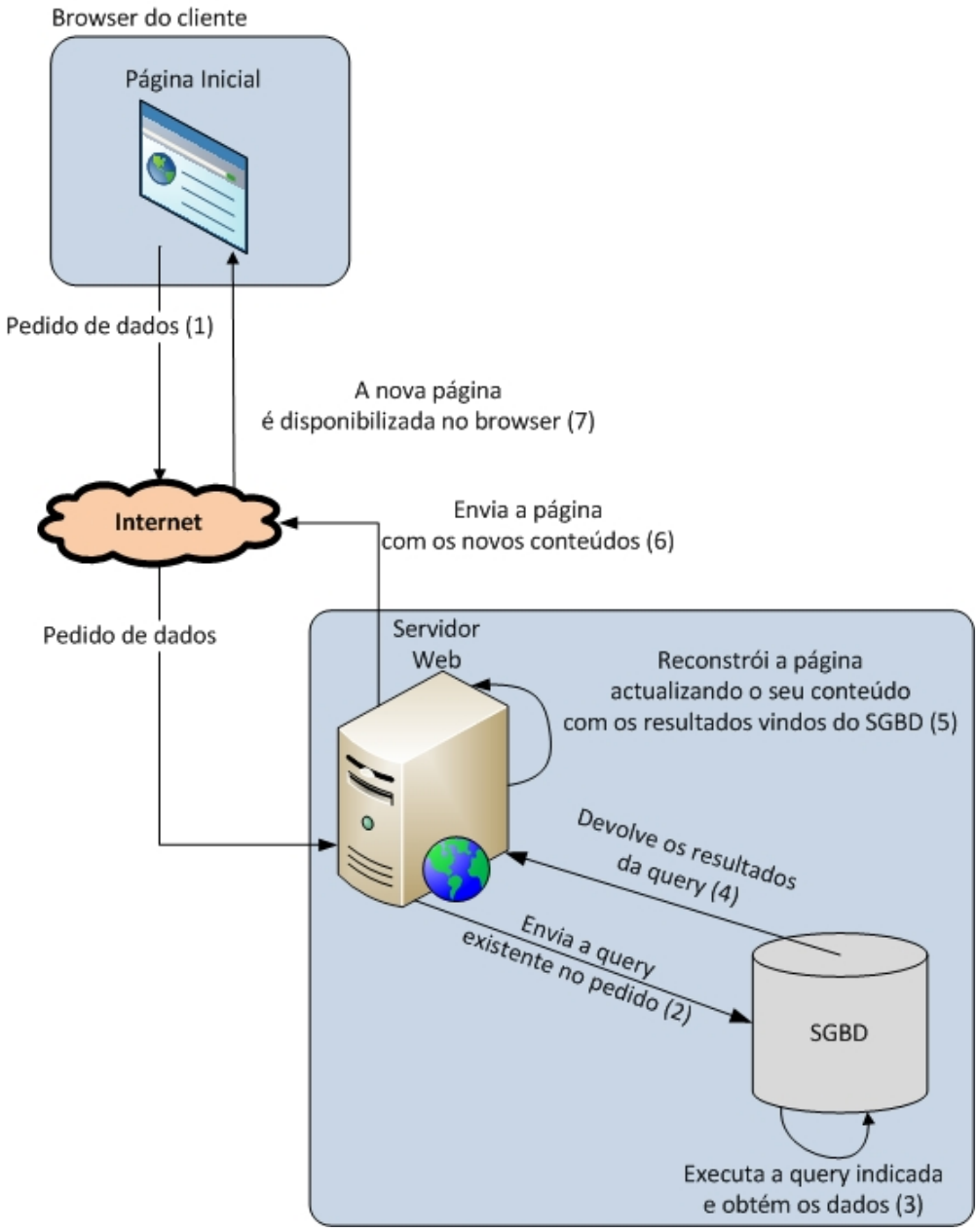


Imagem 21 - Aplicação Web: Funcionamento de uma pesquisa na BD

Neste diagrama convém realçar que o servidor Web, a partir do código ASP .NET, e a partir dos dados provenientes do SGBD, gera código HTML. Esse código é enviado e utilizado pelo *browser* do cliente para disponibilização dos conteúdos respectivos. De realçar também o comportamento apresentado pelo servidor Web quando faz um pedido de dados ao SGBD. Nesta situação o servidor Web passa a funcionar como um cliente, realizando pedidos de dados, e aguardando a resposta do SGBD, para poder processar os resultados obtidos e integrá-los no código ASP.NET correspondente, gerando o código HTML a ser enviado para o *browser* do cliente.

3.5.1. Visualizar informação:

Como já foi mencionado, os módulos pertencentes a esta categoria apresentam um modo de construção e de funcionamento idêntico. O objectivo destes módulos é a apresentação de toda a informação relativa aos Termos, à Ciência, aos prefixos, aos sufixos, aos lexemas armazenados na base de dados.

Todos estes módulos apresentam dois modos de funcionamento. Um dos modos ocorre quando o módulo é chamado directamente, por exemplo a partir do menu, sem apresentar argumentos na *querystring*. Neste modo de funcionamento é apresentada uma lista de toda a informação armazenada na base de dados relativa à função e ao módulo em questão. É possível seleccionar um dos elementos presentes nessa lista e visualizar a informação com mais detalhe. O outro modo de funcionamento ocorre quando, por exemplo, o módulo é chamado a partir da página default.aspx, ou seja, com argumentos na *querystring* que especificam um *id* com que são identificados os dados presentes na base de dados. Para este caso, o módulo disponibiliza logo os dados respectivos.

Embora seja dito que o funcionamento dos módulos é idêntico entre si, a sua complexidade vária com a quantidade dos dados a apresentar. Quando existem poucos dados a visualizar, a construção do módulo é relativamente simples. Nestes casos, os dados provêm no máximo de uma ou mais tabelas. Assim basta utilizar uma simples operação de *select* da ou das tabelas envolvidas para obtenção dos dados.

Um desses casos é o módulo responsável por disponibilizar a informação relativa a uma ciência – o módulo ViewCiencia.aspx.

Este módulo, tal como todos os módulos de visualização, apresenta um conjunto de controlos ASP .NET, definidos no ficheiro de interface, que irão receber os dados respectivos, recolhidos da base dados. Para o módulo em análise, a recolha desses dados é feita numa

operação de *select* na tabela Ciência, existente na base de dados. Os resultados provenientes dessa operação são então utilizados para preencher os controlos ASP definidos na página (neste caso são controlos do tipo *asp:Label*). De forma a ser possível verificar como é que a recolha dos dados de uma ciência é realizada, o código da função correspondente está disponibilizado no Anexo E1. De realçar que este módulo permite ainda a visualização dos subdomínios que, caso existam, pertençam à ciência que esteja a ser visualizada, apresentando toda a informação relativa a eles. A recolha dessa informação é efectuada mais uma vez, através de uma *query* simples de *select* em tudo idêntica a implementada para os dados referentes a uma ciência, alterando naturalmente a tabela e dados a recolher.

Mas nem sempre a quantidade de informação a recolher é pequena e nem sempre se encontra numa só tabela da base de dados. Quando a quantidade de informação que tem de ser disponibilizada é elevada e se encontra espalhada por várias tabelas distintas, a recolha da totalidade da informação, para ser efectuada de uma forma mais eficiente e menos confusa, é realizada através das *Views* que foram definidas na base de dados. A forma como essas Views são implementadas, é exemplificado no Anexo B4. Os módulos que utilizam este mecanismo de recolha de dados são os que se referem à visualização de um prefixo, de um sufixo e de um lexema (respectivamente ViewPrefixo.aspx, ViewSufixo.aspx, ViewLexema.aspx). Tomando como exemplo o módulo ViewPrefixo.aspx, os controlos que irão receber os dados relativos a um prefixo, bem como a sua organização, são exibidos na imagem seguinte:

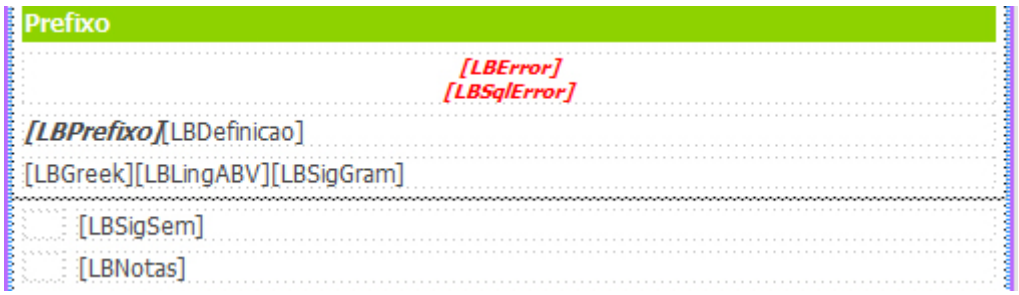


Imagem 22 - Aplicação Web: Estrutura dos controlos para visualização de um prefixo

Estes controlos vão ser preenchidos através da utilização da *View* denominada ViewPrefixo. Como uma *View* apresenta um funcionamento igual ao de uma tabela, a recolha é realizada de uma forma análoga a verificada no módulo de visualização de uma ciência, ou seja, através de uma simples operação de *select* sobre a *View*. Para caso da visualização de um prefixo a implementação da função que utiliza a *View* correspondente está presente no Anexo E2.

Falta mencionar um módulo, que apesar de encaixar nos anteriores, deve ser analisado à parte. Esse módulo é o responsável pela visualização de um termo, o ViewTermo.aspx.

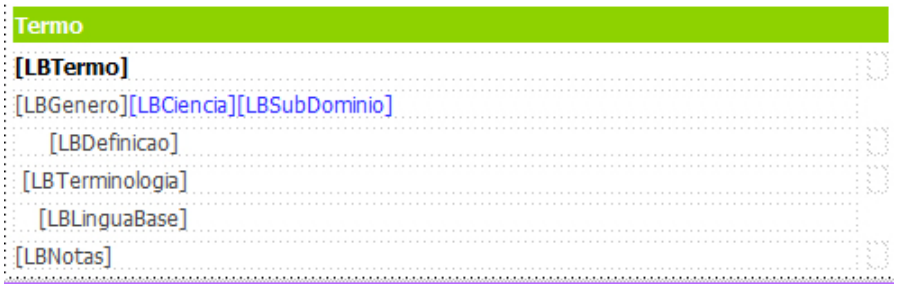


Imagem 23 - Aplicação Web: Estrutura dos controlos para visualização de um termo

Quando comparado com os outros, este módulo é o que recolhe o maior número de dados, como dados provenientes de todas as tabelas. A disponibilização desses dados é realizada em duas fases: preenchimento dos dados directamente relacionados com o termo, como a ciência, a língua, a sua cronologia; e preenchimento dos dados relativos aos seus constituintes morfológicos. A primeira fase é realizada da mesma forma referida para o módulo de visualização de prefixos. Os dados são obtidos utilizando a *View* respectiva e o resultado dessa recolha é utilizado no preenchimento dos controlos respectivos. A única diferença que este módulo apresenta ao nível desse preenchimento é a utilização de controlos do tipo hiperligação (que estão representados a azul) para os dados relacionados com a ciência e o subdomínio científico, que permitem uma visualização mais detalhada da informação correspondente.

O preenchimento dos constituintes morfológicos está separado do preenchimento anterior, porque é necessário processar a informação respectiva que é recolhida da base de dados. Esse processamento está relacionado com a ordem de visualização pela qual cada um dos constituintes morfológicos tem de ser exibido. Para obter tal ordem é utilizada a tabela definida na base de dados TermoDisplay. E é a partir dessa ordem que os constituintes morfológicos são disponibilizados pela ordem correcta. Mas para compreender melhor este mecanismo, deve ser analisado um exemplo. Supondo que a informação da ordem recolhida da tabela TermoDisplay é a seguinte:

P1;L1;L2;S1;

Onde cada letra indica o tipo de constituinte morfológico (P – prefixo, S – sufixo e L – lexema) e o número indica posição de cada um deles nas tabelas que os relacionam com o termo respectivo. Percorrendo sequencialmente esta informação, e utilizando o caracter ponto e virgula

como separador entre os constituintes morfológicos, a sua obtenção ordenada é realizada da seguinte forma:

1º. O primeiro valor obtido é P1, o que indica que o primeiro constituinte morfológico a ser disponibilizado é um prefixo, sendo necessário obter então o identificador único desse prefixo, na tabela Prefixos.

2º. O identificador único do prefixo é obtido através da tabela que relaciona um termo com um prefixo, a tabela TermoPrefixo, utilizando uma simples *query* de *select* nessa tabela e especificando o identificador único do termo correspondente bem como o número que indica a posição.

3º. A partir do identificador único do prefixo basta obter os dados do prefixo correspondente, da mesma forma que já foi mencionada para o módulo de visualização de em prefixo, através da View ViewPrefixo.

4º. A partir do resultado anterior, os controlos correspondentes são preenchidos como os valores respectivos.

5º. O constituinte morfológico seguinte é o L1, neste caso um Lexema. A partir daqui o funcionamento repete-se de uma forma análoga até ao último constituinte morfológico, alterando naturalmente as tabelas de onde a informação é recolhida: para um lexema, as tabelas TermoLexema e Lexema; para um sufixo, as tabelas TermoSufixo e Sufixo. Para a disponibilização são utilizadas as *Views* correspondentes a ViewLexema e a ViewSufixo.

Este mecanismo é realizado por um grupo de funções definidas neste módulo. No Anexo E3 é exibida a função que disponibiliza todos os constituintes morfológicos de uma forma ordenada, no Anexo E4 a função responsável pela obtenção do identificador único de um prefixo. A partir do identificador único os dados do prefixo respectivo são obtidos através da *View* respectiva. Para os restantes constituintes morfológicos, de um lexema e de um sufixo, a implementação destas funções é análoga.

De realçar, que todos os constituintes morfológicos são colocados no mesmo controlo *asp:label*, mas de forma separada, utilizando código HTML. Essa separação é realizada através do comando HTML <a ref> … que cria uma hiperligação. Assim cada constituinte morfológico que um termo apresente, aparecerá em hiperligação, que ao ser clicado, direcciona a aplicação para a visualização da informação do constituinte morfológico respectivo.

Para finalizar, uma palavra sobre a arquitectura que os módulos de visualização seguem. A arquitectura seguida por estes módulos é idêntica à apresentada na Imagem 21. De notar que, mesmo nos casos em que a visualização da informação utiliza as *Views* definidas na base de dados, e uma vez que essas *Views* possuem um funcionamento igual a uma tabela normal, o acesso a elas é realizado por uma simples *query* de *select*.

3.5.2. Inserir informação

Os módulos de inserção de dados apresentam uma construção, e um funcionamento mais complexo que os módulos de visualização. Isso deve-se ao facto de estes módulos serem responsáveis pela inserção na base de dados e para que tal ocorra sem problemas é necessário proceder à validação dos dados antes de os submeter para a base de dados. É ainda necessário apresentar mecanismos de detecção de falhas ocorridas no servidor de base de dados.

Validação dos dados:

A validação e os mecanismos de detecção de falhas englobam o preenchimento de campos obrigatórios, verificando se o tamanho dos dados inseridos não é maior do que está configurado nas tabelas da base dados e se não se está a tentar inserir algo que já se encontra armazenado ou outro tipo de erro. A validação e o respectivo alerta para o utilizador são realizados sempre da mesma forma em todos os módulos.

A validação é realizada por um conjunto de dois controlos: o *asp:RequiredFieldValidator* e *asp:RegularExpressionValidator*.

O *asp:RequiredFieldValidator* é utilizado para verificar se o controlo está ou não preenchido, devolvendo uma mensagem de erro em caso negativo. Este controlo é utilizado para caixas de textos que recebem dados que são obrigatórios. A seguir é apresentado um exemplo da respectiva definição:

```
<asp:RequiredFieldValidator ID="ValidatorLexema" runat="server" ControlToValidate="TBLexema"
ValidationGroup="LexemaVal" ErrorMessage="Complete o Lexema" Display="Dynamic"> *
</asp:RequiredFieldValidator>
```

Para as caixas de texto simples, a validação do tamanho dos dados inseridos é validado no próprio controlo da caixa de texto, através do atributo *MaxLength*. Mas o mesmo não acontece

para o tipo de caixa de texto com multi-linhas. Neste caso, o atributo *MaxLength* não funciona e para que seja possível validar o tamanho dos dados, ou seja o número de caracteres, é necessário utilizar o controlo *asp:RegularExpressionValidator*. Este tipo de controlo é utilizado para determinar se o valor de um dado controlo de *input* respeita uma dada expressão regular. Da mesma forma, o controlo permite verificar sequências de caracteres que tenham uma estrutura fixa e definida, como endereços de correio electrónico, números de telemóvel. No caso do DICITER, é utilizado para verificar o número de caracteres que foram escritos no controlo. Para tal, recorre-se à expressão regular ^[\s\S]{0,250}$, onde o acento circunflexo significa o princípio de uma *string*; o [\s\S] representa o conjunto de caracteres a verificar, o \s significa todos os espaços em branco e o \S o inverso, ou seja, o conjunto vai conter todos os caracteres e todos os espaços em branco. Os números dentro de chavetas definem o número mínimo e máximo de repetições definidas pelo conjunto anterior. Por fim, o caracter dólar indica o fim da *string*. De seguida, é possível ver um exemplo da definição do controlo mencionado.

```
<asp:RegularExpressionValidator ID="rgConclusionValidator2" ControlToValidate="TBConceito"
ErrorMessage="Conceito: maximo 50 Caracteres" ValidationExpression="^[\s\S]{0,50}$" runat="server"
Display="Dynamic" SetFocusOnError="true" ValidationGroup="LexemaVal">Max 50 Caracteres
</asp:RegularExpressionValidator>
```

Alertas de erros:

Caso ocorram erros na validação, a aplicação produz um alerta para o utilizador. É criada uma janela de *popup* que avisa o utilizador do tipo e da localização do erro que ocorreu. Essa janela pode ser criada de duas formas, dependendo do erro. Se o erro é proveniente dos controlos de validação anteriormente definidos, como uma caixa de texto obrigatória não preenchida, ou o limite de caracteres nas caixas de textos multi-linha foi ultrapassado, é utilizado o controlo *asp:ValidationSummary*.

```
<asp:ValidationSummary ID="ValidationSummary2" runat="server" ValidationGroup="LexemaVal"
ShowMessageBox="True" ShowSummary="False" HeaderText="Atenção:" />
```

Este controlo, como o próprio nome indica, cria um sumário com todos os erros que os controlos de validação detectaram. Assim é possível criar o alerta para o utilizador num só sítio. No DICITER esse alerta é realizado através de um *popup*, para além de aparecer um asterisco ao lado de cada controlo que provocou o alerta de validação.

Existem outros tipos de erros ou falhas em que é necessário alertar o utilizador, como é o caso da inserção com sucesso ou não. Para tal foi utilizado javascript para criar uma janela de *popup* que alerta o utilizador se a inserção foi realizada com sucesso ou, caso contrário, o erro ocorrido. O alerta javascript foi criado numa classe separada (Alert.cs, colocada na pasta App_Code) que permite a sua fácil reutilização, sendo necessário apenas invocar a função definida na classe anterior, passando a mensagem de erro que deve de ser apresentada, como verificado no exemplo seguinte:

```
Alert.Show("Inserção da ciencia " + CienciaTB.Text + "foi efectuada!");
```

Está função irá gerar um *popup* do género exibido na figura seguinte:

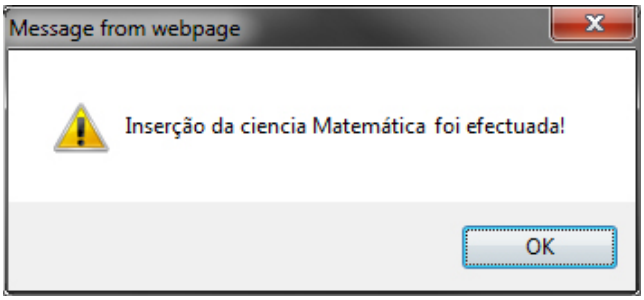


Imagem 24 - Aplicação Web: Alerta Javascript

O código completo da classe Alert.cs encontra-se no Anexo F1.

Inserção dos dados:

Por uma questão de conformidade ao longo de toda a aplicação os módulos de inserção dos dados apresentam o mesmo aspecto e implementação. Eles apresentam vários controlos que correspondem aos dados relativos à informação a inserir como uma ciência, um subdomínio científico, um prefixo, um lexema, um sufixo. O módulo para inserção de termos, como apresenta uma grande quantidade de dados a inserir na base de dados, apresenta um aspecto e implementação um pouco diferente dos restantes módulos, apresentando algumas características próprias que serão analisadas mais à frente.

Existem várias possibilidades para realizar a inserção dos dados, dependendo da quantidade de dados que é necessária inserir bem como das tabelas por onde eles se encontram espalhados. Quando a quantidade dos dados a inserir é pequena e se encontram numa só tabela que não referencia ou não é referenciada por outras tabelas, a inserção dos dados pode ser

realizada por um controlo do ASP .NET. Esse controlo é o *asp:SqlDataSource*. Este permite o acesso a dados contidos numa base de dados relacional, incluindo o Microsoft SQL Server e a base de dados Oracle. É possível a sua utilização com outros controlos que permitem a visualização de dados como o *asp:GridView*. Também é possível especificar uma operação de inserção responsável pela inserção dos dados.

Este modo de inserção é utilizado pelo módulo InsertCiencia.aspx, que é o mais simples, apresentando dados a inserir numa só tabela. Os controlos, que irão receber os dados a serem inseridos podem ser verificados na seguinte figura:

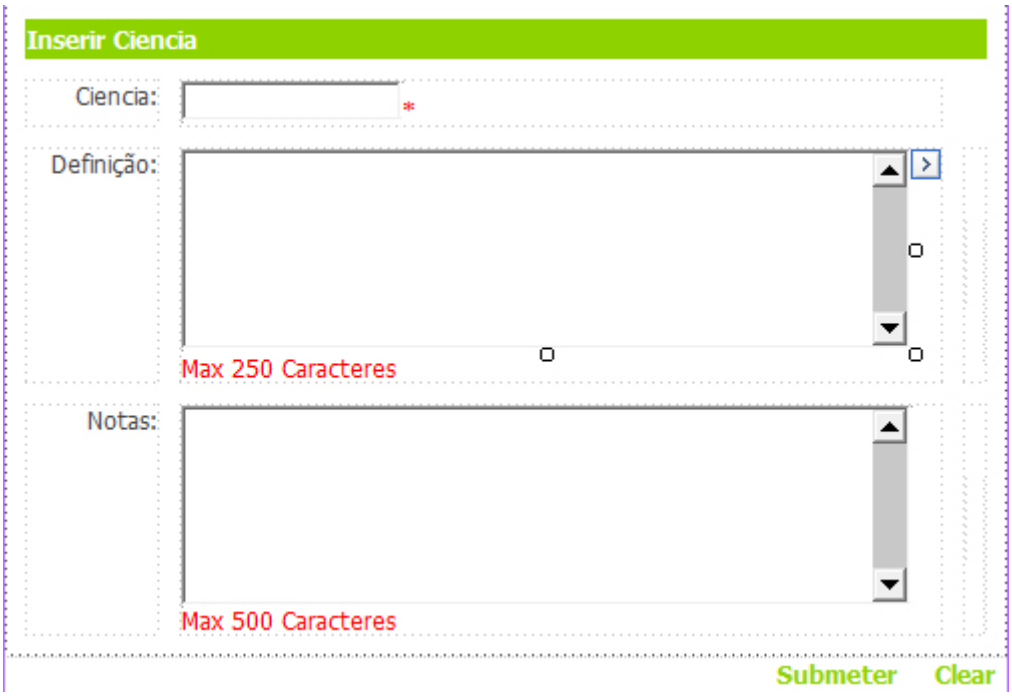


Imagem 25 - Aplicação Web: Estrutura dos controlos para inserção de uma ciência

Os controlos que são visualizados acima vão ser utilizados para inserção dos dados, através da sua atribuição às variáveis criadas no comando de *insert* do controlo *asp:SqlDataSource*.

Para realizar esta operação de inserção basta invocar o controlo anterior, especificando a operação de *insert* presente no controlo *asp:SqlDataSource*, através de uma função que se encontra associada ao evento *click* do botão submeter que está definido no módulo (a função é executada depois do botão ser clicado). Esta função, tanto em caso de sucesso ou de erro, notifica o utilizador através de um *popup* do tipo mencionado anteriormente. De realçar o caso particular que ocorre quando se tenta inserir uma ciência já existente na base de dados. Como foi mencionado no capítulo dedicado à construção da base de dados, a tabela ciência apresenta um

índice do tipo único que impede a repetição de valores (o código de definição desse índice e do todos os configurados na base de dados pode ser consultado no Anexo B2). Quando isso acontece, o SQL Server gera um erro (cada erro que pode ocorrer no SQL Server apresenta um número de identificação). Para capturar o erro específico de violação do índice único, basta verificar o número de identificação correspondente e alertar o utilizador para o facto de a ciência já existir na base de dados. O código respectivo da função acabada de mencionar e do controlo *asp:SqlDataSource* que insere os dados está presente no Anexo H1.

Quando a quantidade de dados a inserir é mais elevada e estão dispersos por várias tabelas da base de dados, são utilizados os *stored procedures* de inserção criados na base de dados. Os módulos que inserem informação relativa a um subdomínio científico, a um prefixo, a um sufixo ou a um termo apresentam uma forma de inserção semelhante. Estes apresentam vários controlos do tipo *asp:TextBox* que vão receber os dados que depois serão passados para o *stored procedure* correspondente. A imagem seguinte exibe os controlos que são colocados na área central pelo módulo de inserção de prefixos – InsertPrefixo.aspx.

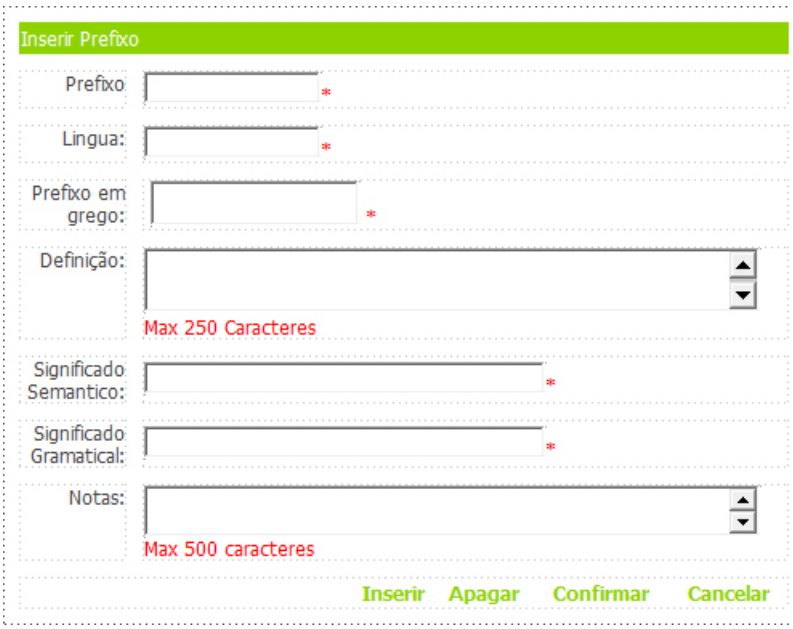


Imagem 26 - Aplicação Web: Estrutura dos controlos para inserção de um prefixo

De ter em conta que este módulo, e os outros deste género, antes de a inserção ser realizada, apresenta uma página onde é pedida a confirmação dos dados a inserir. Depois de terem sido confirmados, a inserção é realizada.

Para a operação de inserção de, por exemplo, um prefixo é necessário inserir os dados nas tabelas que vão ser referenciadas pela tabela prefixo ou nas que vão referenciar a tabela prefixo.

Os dados referentes às tabelas língua e dos significados semânticos e gramaticais têm de ser inseridos primeiro, caso não existam. Estas inserções são simples, utilizando para o efeito, o código T-SQL. De notar que durante a própria inserção o *id* atribuído, é obtido para poder ser passado para o *stored procedure* de inserção respectivo (neste caso o InsertPrefixo). Caso esses dados já existam na base de dados o *id* é obtido (através dos stored procedures que verificam a existência dos dados), e passado para o *stored procedure*.

Só a partir daí é que é possível realizar a inserção do prefixo respectivo. A inserção de todos os dados é realizada de uma forma encadeada. Assim, caso ocorra algum erro durante a operação de inserção, a detecção da falha é mais fácil. Finalmente a inserção dos dados na tabela Prefixo é realizada, utilizando o *stored procedure InsertPrefixo* configurado na base dados e passando para os seus parâmetros, os dados respectivos. No Anexo B3 é apresentado a forma como é realizada a implementação dos *stored procedures* através da apresentação de código de alguns deles. E como exemplo, o código responsável pela inserção de um prefixo (que usa um desses *stored procedures*) é apresentado no Anexo H2, onde é possivel ver como um dado *stored procedure* é invocado,e os seu valor de retorno obtido.

Arquitectura funcional:

Em relação à arquitectura seguida por estes módulos, convém mencionar o funcionamento dos módulos que usam os *stored procedures*, apesar de esse funcionamento ser em tudo idêntido ao apresentado para uma pesquisa. A imagem seguinte exibe o funcionamento do módulo de inserção de prefixos, que serve de exemplo para todos os módulos que utilizam os *stored procedures*.

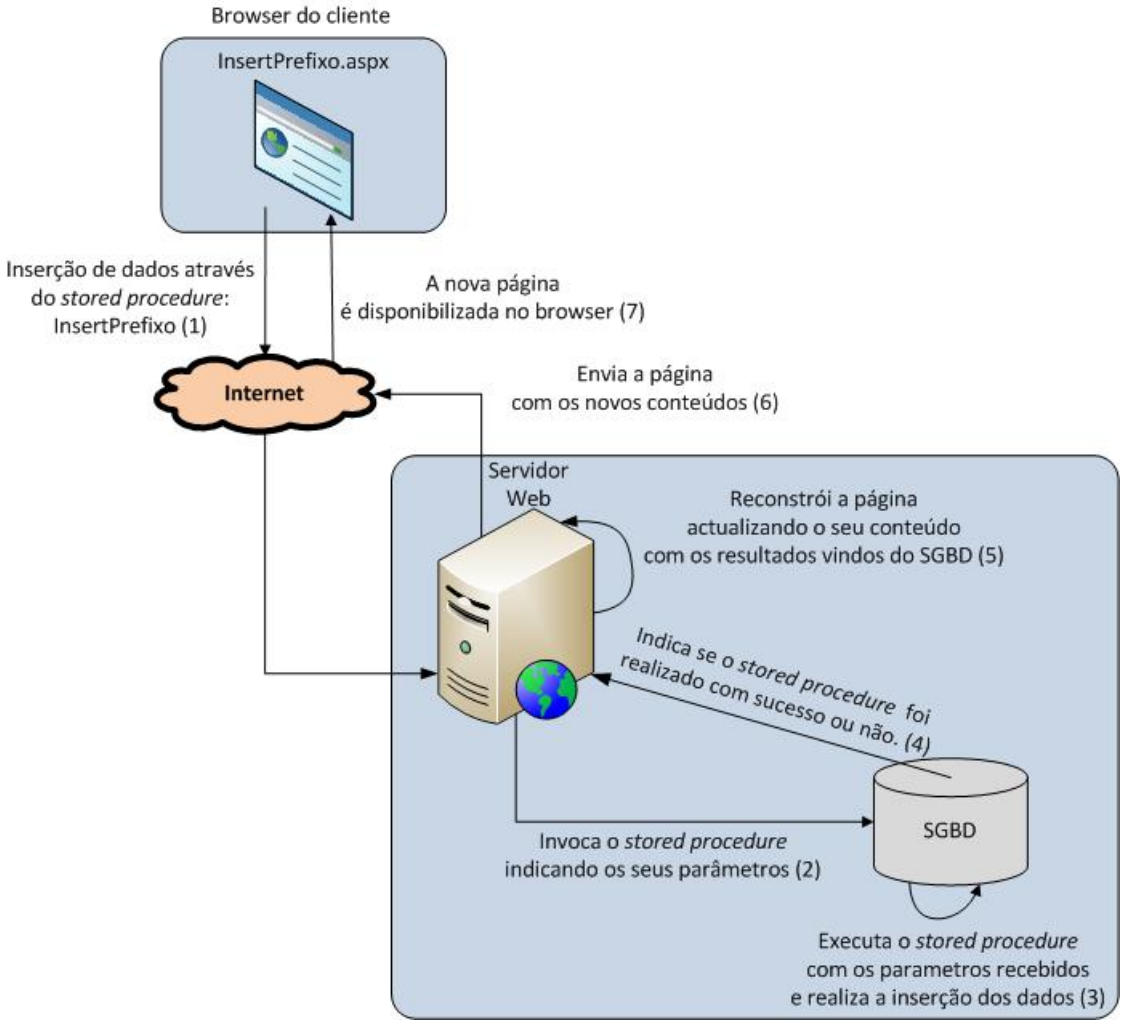


Imagem 27 - Aplicação Web: Arquitectura da inserção na BD através de *stored procedures*

É necessário explicar alguns aspectos apresentados na figura anterior. Como é possivel verificar, o funcionamento é de alguma forma idêntico ao apresentado pela pesquisa de dados e pelos módulos de visualização. Mas neste caso é invocado um *stored procedure* que realiza a operação desejada, enviando todos os seus parâmetros de entrada. Como indicado na imagem, o SGBD notifica o servidor Web se essa operação foi executada com sucesso ou não. De realçar que

quando a execução é realizada com sucesso, se o *stored procedure* implementar valor de retorno, esse valor também é enviado para o servidor Web de forma a ser utilizado na reconstrução da página a disponibilizar no cliente. É a partir deste valor de retorno que é possível a notificação de sucesso da operação de inserção no *browser* do cliente. Essa notificação, que já foi analisada, é criada pelo *browser* do cliente através de uma função JavaScript.

Suporte para caracteres gregos:

Para finalizar a análise dos módulos de inserção, nos módulos relativos aos constituintes morfológicos existe um caso particular referente a inserção de caracteres gregos na base de dados. Para que a aplicação suporte caracteres gregos (com os vários tipos de acentuação) é necessário um mecanismo que converta os caracteres do teclado no caracter grego correspondente, bem como atribuir as teclas para os acentos. Isto é conseguido através da utilização de uma função escrita em javascript. Essa função é associada à caixa de texto, destinada a receber dados em grego, de forma que quando esta recebe alguma letra, esta é imediatamente convertida para o caracter grego correspondente (o mesmo irá ocorrer para os acentos). Apesar da utilização deste mecanismo, também é possível utilizar o método copiar e colar, copiando os dados de um documento e colando-os depois na caixa de texto. O código da função javascript faz a troca dos caracteres, como é possível verificar no Anexo F2.

A construção dessa caixa de texto é simples, bastando utilizar um controlo HTML do tipo input. Este exemplo foi retirado do módulo de inserção de prefixos, que é sempre realizado da mesma forma, qualquer que seja o módulo.

```
<input id="TBPrefixoGreek" type="text" runat="server"  class="textboxgreek" />
```

É utilizado um estilo CSS para definir o tipo de letra utilizado. Este tipo de letra é diferente do utilizado no resto da aplicação, porque é necessário um tipo de letra que suporte todos os caracteres gregos. O tipo de letra utilizado foi o Palatino Linotype. Por fim é necessário atribuir a função javascript ao controlo.

De realçar que esta função javascript, como todas as funções javascript, é executada no *browser* do cliente, sem influência do servidor web. Como referido anteriormente, todos os *browsers* actuais processam código javascript, que é encapsulado no código HTML de uma página web.

A forma como esse processamento é realizado, está ilustrada de maneira simples na imagem seguinte:

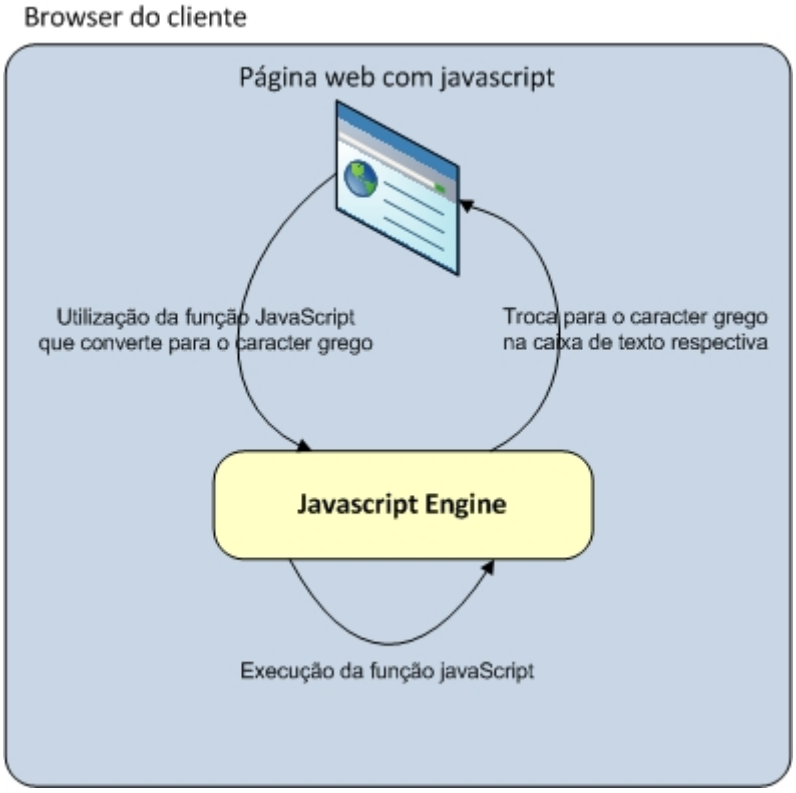


Imagem 28 - Aplicação Web: Funcionamento do Javascript

A conversão para o caracter grego é realizada de uma forma automática e quase imediata. Uma vez mais, o processamento das funções javascript é realizado no cliente, mais especificamente pelo *browser*.

3.5.6. Utilização de ASP .NET AJAX

Antes de analisar as características próprias do módulo para inserir termos, é necessário falar dos controlos ASP .NET AJAX utilizados para tornar a operação de inserção menos complicada e eficiente. De notar que o ASP .NET AJAX foi utilizado em todos os módulos de inserção. Basicamente são utilizados os mesmos controlos e da mesma forma ao longo de todos os módulos, sendo que o que apresenta controlos únicos é o modulo de inserção de um termo.

ASP .NET AJAX FilteredTextBox:

Na maior parte das caixas de textos presentes no site foi utilizado o controlo FilteredTextBox que impede a inserção de caracteres considerados inválidos na caixa de texto. Dependendo do objectivo da caixa de texto, este controlo permite que, por exemplo, só sejam inseridos números ou letras do alfabeto (com a possibilidade de distinção entre maiúsculas e minúsculas) ou seja definido um conjunto de caracteres. Um exemplo da definição de um controlo FilteredTextBox é apresentado a seguir:

```
<cc1:FilteredTextBoxExtender ID="CienciaTB_FilteredTextBoxExtender" runat="server"
Enabled="True" TargetControlID="CienciaTB" InvalidChars="%&/|\<>" FilterMode="InvalidChars">
</cc1:FilteredTextBoxExtender>
```

ASP .NET AJAX AutocompleteExtender:

No módulos de inserção de um termo, de um prefixo, de um lexema e de um sufixo é utilizado um controlo AJAX oriundo da ASP.NET AJAX Control Toolkit. Este controlo é associado às caixas de textos que representam os dados referentes à língua, ao significado semântico e gramatical, e à classe gramatical. Esse controlo é denominado *AutoCompleteExtender*. Este controlo estende as caixas de texto do ASP de forma a apresentarem preenchimento automático dos dados que elas representam. Esse preenchimento é realizado com dados já existentes na base de dados, dessa forma é possível escolher um que já esteja armazenado e evitar erros de escrita por parte do utilizador, evitando o armazenamento de dados desnecessários e incorrectos na base de dados. Um exemplo desse controlo ASP.NET AJAX é apresentado a seguir:

```
<cc1:AutoCompleteExtender ID="ACPrefixo" runat="server" MinimumPrefixLength="1"
ServiceMethod="GetLingua" ServicePath="~/WebService.asmx" TargetControlID="TBLinguaPrefixo"
CompletionInterval="1"> </cc1:AutoCompleteExtender>
```

A recolha dos dados, para o preenchimento automático é realizada através de um *WebService*. A definição do *WebService* está presente num ficheiro, denominado WebService.asmx e localizado na raiz da aplicação, onde é indicada a linguagem de programação em que o WebService foi construído bem como a sua localização.

```
<%@ WebService Language="C#" CodeBehind="~/App_Code/WebService.cs" Class="WebService" %>
```

O código funcional está definido num ficheiro WebService.cs, que não é nada mais, do que uma classe C#. Nesta classe estão presentes todas as funções implementadas para o preenchimento automático de caixas de texto. A respectiva construção e implementação são análogas. O código exemplificativo dessa classe pode ser observado no Anexo I1.

ASP .NET AJAX TabContainer:

O módulo InsertTermo.aspx apresenta a possibilidade de inserção de todos os dados que existem na base de dados, associando-os a um novo termo. Naturalmente permite também a utilização de dados já existentes na base de dados que fazem parte do novo termo que irá ser inserido.

Devido a esta grande quantidade de dados, este módulo apresenta uma separação em *tabs*. Cada *tab* vai agrupar um conjunto de dados de forma a criar um aspecto e usabilidade menos confusa.

Os dados foram separados em cinco grupos: apresentação de controlos para os dados como o nome, a definição, a língua, a classe gramatical e o género; apresentação de dados que estão relacionados com a cronologia do termo e com o tipo de termo, o ano, o século e a fonte de onde se retirou esta informação; escolha ou inserção dos constituintes morfológicos de uma forma ordenada; inserção de notas relativas a um termo. Por fim, foi criada uma tab denominada terminar que vai permitir a confirmação da inserção do termo, utilizando uma estrutura idêntica da utilizada para a visualização do termo.

Estas *tabs* são criadas através de um controlo disponibilizado pelo ASP .NET AJAX Control Toolkit denominado de *TabContainer* que permite a definição de múltiplas *tabs*. Um exemplo da definição do controlo pode ser observado a seguir:

```
<cc1:TabContainer ID="TabContainer1" runat="server" Visible=”true">
<cc1:TabPanel ID="TAB1" HeaderText="Geral" runat="server">
        <ContentTemplate>(...) </ContentTemplate>
</cc1:TabPanel>
<cc1:TabPanel ID="TAB2" HeaderText="Ciencia" runat="server">
        <ContentTemplate> (...) </ContentTemplate>
</cc1:TabPanel>
(…)
```

É dentro destes blocos de código (*<ContentTemplate>(…) </ContentTemplate>*) que são definidos todos os controlos ASP .NET que fazem parte do *tab* correspondente.

Tal como outros controlos, é possível associar eventos ao controlo TabContainer para realizar alguma operação específica. Neste caso foi configurado um evento para quando ocorre alteração da *tab* seleccionada. Este evento vai permitir a validação dos controlos necessários e correspondente alerta em caso de erro, quando a última *tab* é seleccionada.

Arquitectura funcional:

A forma de funcionamento de um controlo AJAX já foi referida no subcapítulo 3.4, dedicado à apresentação da arquitectura global do sistema. Na seguinte imagem é ilustrado o funcionamento simplificado de um controlo ASP .NET AJAX que necessita de dados provenientes da base de dados, como ocorre com o controlo ASP .NET AJAX AutocompleteExtender, analisado anteriormente.

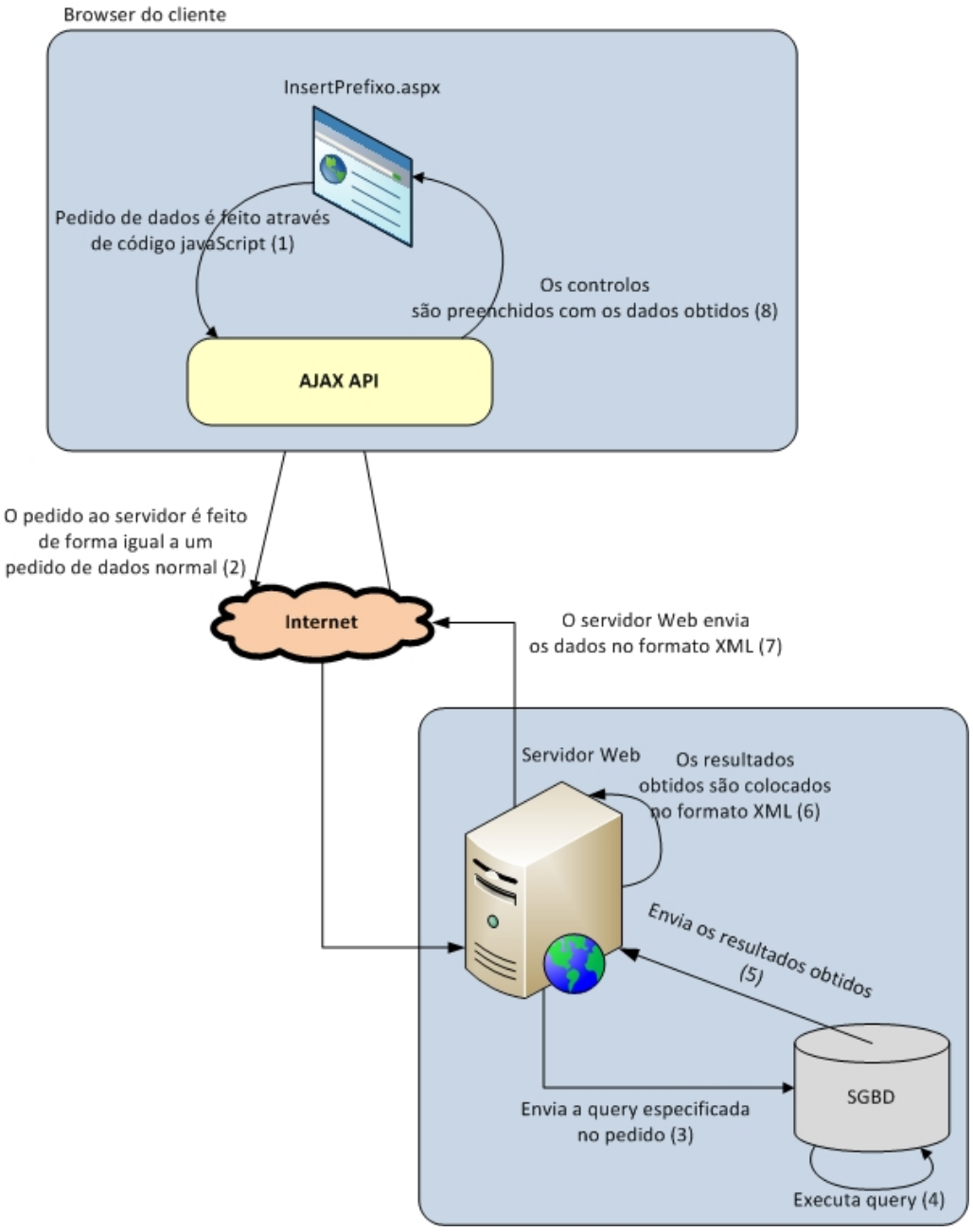


Imagem 29 - Aplicação Web: Funcionamento do AJAX

Na imagem convém realçar que, graças ao ASP .NET AJAX, o pedido de dados realizado pelo *browser* do cliente é totalmente transparente para o utilizador. Graças ao seu funcionamento assíncrono, o utilizador não nota que estão a ser enviados e recebidos dados ao servidor. Os dados que são envidados no formato XML por parte do servidor Web (7) para o *browser* do cliente, são processados pelo AJAX API para a geração de código HTML, que irá ser utilizado no funcionamento do controlo AJAX respectivo. Esta forma de funcionamento do ASP.NET AJAX providencia a criação de páginas Web ainda mais interactivas e funcionais e com grande acessibilidade.

3.5.3. Aspectos particulares do módulo de inserir termos:

O módulo de inserção de termos, apresenta um grande número de dados a inserir. Para além disso, existem dados que para serem obtidos e inseridos necessitam de tratamento especial. É o caso da existência de múltiplas definições, e dos constituintes morfológicos.

Para o sistema suportar a inserção de várias definições temos de criar controlos de uma forma dinâmica. Como já foi referido, esses controlos vão ser colocados no controlo *asp:Panel*, definido na página. Esses controlos irão receber os dados, dependendo da quantidade de definições estabelecida. Para cada definição são necessários dois controlos do tipo caixa de texto: uma para a classe gramatical, uma vez que diferentes definições podem corresponder a diferentes classes gramaticais (o preenchimento é opcional); e outra para a definição.

O número de definições é estabelecido pelo utilizador, numa caixa de texto específica. Esse valor vai indicar a quantidade de caixas de textos a serem criadas dinamicamente. Mas antes de criar as caixas de texto propriamente ditas, como elas são dinâmicas, é necessário criar algo para as armazenar. Isso é feito através de variáveis do tipo *list* que vão armazenar objectos do tipo *TextBox*, que correspondem às caixas de texto. A partir do número de caixas de texto necessárias e das variáveis anteriormente definidas, é possível preencher o controlo *asp:Panel* com o número de caixas de texto desejado, bem como através de código HTML, definir uma estrutura para as apresentar.

Com a utilização de controlos dinâmicos é necessário ter atenção o mecanismo de *postback* que uma página ASP .NET apresenta. O mecanismo de *postback*, ocorre quando a pagina já inteiramente disponível no cliente, é novamente enviada para o servidor para processamento de controlos e ou dados. No fim desse processamento, o servidor volta a exibir a mesma página com as alterações resultantes. Este comportamento ocorre, por exemplo, quando é seleccionada

outra *tab*. Devido a este mecanismo, é necessário recriar novamente os controlos que foram gerados dinamicamente. Se não forem recriados estes controlos, os seus dados e atributos, não vão estar disponíveis. A reconstrução dos controlos dinâmicos é realizada por uma função, invocada caso o mecanismo de *postback* ocorra, reconstruindo as caixas de texto da mesma forma que na criação. A captura de um *postback* é realizada facilmente pela utilização de uma variável de sistema booleana, denominada *IsPostback* que indica se a página está a ser lida em resposta a um *postback* do cliente, ou se está a ser lida pela primeira vez. Essa verificação é realizada durante a leitura da página, ou seja no *Page_Load*.

Depois de os controlos dinâmicos estarem definidos, e a respectiva reconstrução estar assegurada, após um postback, é necessário um mecanismo que recolha os dados de todos eles, de forma a armazená-los no campo respectivo da tabela Termo. Como é fácil de verificar temos dados provenientes de vários controlos e um só sítio para os armazenar. Para ser possível armazenar todos os dados num só campo, é necessário concatená-los com uma estrutura predefinida, para uma posterior recuperação e utilização. A estrutura definida é a seguinte, utilizando o ponto vírgula como separador.

(*classe gramatical*) *definição nº 1;* 2. (*classe gramatical*) *definição nº 2 …*

As funções e o respectivo código das funções responsáveis por estes mecanismos de construção dos controlos dinâmicos e respectiva concatenação de toda a informação estão disponíveis para análise no Anexo J1.

Para terminar a análise da construção da aplicação Web, além desta necessidade de inserir várias definições distintas, também é necessário armazenar a ordem dos constituintes morfológicos de um dado termo. Para tal, a *tab* correspondente (denominada Morfologia) apresenta controlos que possibilitam essa associação ordenada, apresentando um controlo onde é possível escolher o tipo de constituinte: um prefixo, ou um lexema, ou um sufixo.

A imagem seguinte apresenta a estrutura dos controlos para inserção dos constituintes morfológicos:

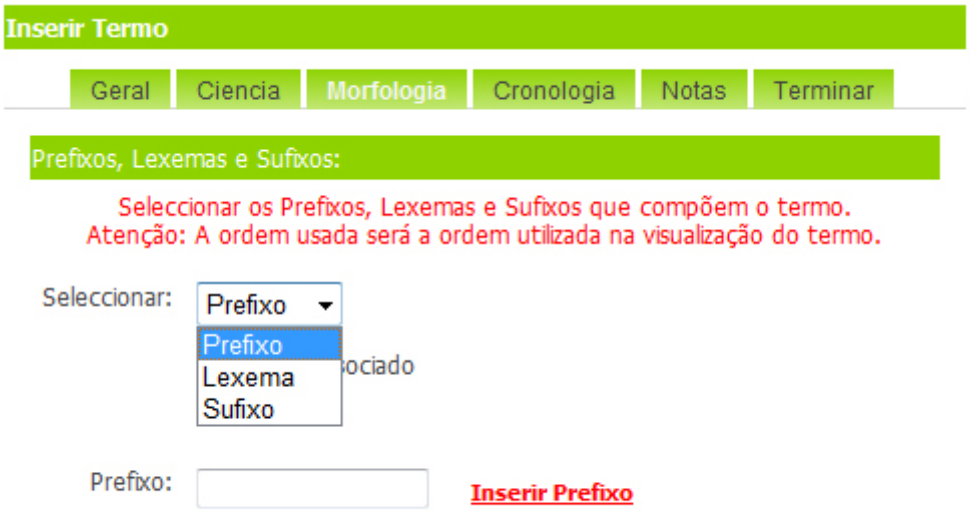


Imagem 30- Aplicação Web: Tab para a associação dos constituintes morfológicos

A partir daí só é necessário realizar uma procura na caixa de texto e com a ajuda do preenchimento automático, caso seja um constituinte que já exista na base de dados, associá-lo ao termo, clicando no botão respectivo. Caso o constituinte morfológico não exista, é possível utilizar a mesma caixa de texto, surgindo para este caso, todos os controlos necessários para o novo constituinte morfológico, com a mesma estrutura utilizada nos módulos de inserção já referidos. A associação ordenada resultante, é armazenada numa variável de sessão, utilizando uma estrutura idêntica à analisada na visualização da informação, que em vez de indicar a posição relativa do constituinte, indica o identificador único respectivo.

Após o processamento da informação contida nessa variável de sessão são retirados e obtidos os dados que vão ser inseridos nas tabelas que relacionam um termo com os seus constituintes morfológicos. É criada também a informação da ordem, em que os constituintes morfológicos devem ser apresentados e que irá ser armazenada na tabela TermoDisplay. A junção da informação que é armazenada nestas tabelas; a TermoDisplay, a TermoPrefixo, a TermoLexema e a TermoSufixo; é depois utilizada para a recuperação da ordem, apresentada pelos constituintes morfológicos.

4. Conclusões e melhoramentos futuros:

Actualmente, com a existência da Internet e de tecnologias que permitem a integração de informação em sistemas centrados na Web, faz todo o sentido a construção de sistemas de informação com enorme relevância no estudo e análise dos mais variados tipos de informação. Devido a utilização da Internet como meio de disponibilização, esses sistemas apresentam uma acessibilidade fora do alcance das tradicionais aplicações *Desktop*, uma vez que permitem o acesso à escala global. Esta enorme acessibilidade permite a utilização do sistema em qualquer local do planeta (desde que exista acesso à Internet) o que aumenta o número de utilizadores que podem tirar partido ou mesmo contribuir, para o estudo e análise da informação apresentada por esses sistemas.

Nesta dissertação tentou apresentar-se uma proposta de um sistema de informação que tivesse todas as vantagens apresentadas anteriormente: a criação de um sistema de informação que esteja ao alcance do maior número de utilizadores possível, que disponibilize de uma forma rápida e de fácil compreensão a informação armazenada e que permita contribuições por parte dos utilizadores que usam o sistema. O sistema proposto, o DiCiTer, implementado com base numa arquitectura centrada na Web, engloba e levou ao estudo de várias tecnologias existentes como o HTML, o DOM, o XML, o Javascript, o CSS, o ASP.NET, os *Web Services*, o Visual Studio 2008, o ASP.NET AJAX e o Microsoft SQL Server 2005.

Com a conjugação de todas estas tecnologias foi possível a construção de um sistema de informação acessível que integra informação referente à terminologia de termos provenientes das várias áreas científicas. O DiCiTer, é uma plataforma de trabalho não só para profissionais ligados às áreas do estudo das línguas, mas para todos os profissionais das áreas científicas que são abordadas. É uma ferramenta que providencia grandes avanços no estudo e análise para a área da terminologia científica, apresentando uma enorme variedade de dados que caracterizam cada termo científico estudado, englobando informação que vai desde a classificação gramatical, da área científica e do histórico, até aos constituintes morfológicos (Prefixos, Lexemas, Sufixos). E por ser um sistema centrado na Web, acessível a partir de qualquer localização onde haja acesso à Internet, é dotado de uma acessibilidade notável que torna o acesso à informação mais fácil, permitindo contribuições no crescimento da informação armazenada de uma forma célere e que potencia em larga escala o objectivo central do sistema desenvolvido, o estudo e análise da terminologia de termos científicos.

O futuro do sistema está dependente do estudo de áreas onde possa ser utilizado facilmente. Pode ser alargado para suportar aspectos mais específicos das áreas científicas, como apresentar suporte para imagens relativas a cada termo apresentado. Também poderá ser considerado o suporte para a apresentação de referências bibliográficas ou hiperligações para *sites* de Internet onde cada termo, ou o assunto a que o termo se refere, seja mencionado ou mesmo estudado. De uma forma mais abrangente e global, poderá ser aperfeiçoado o sistema estatístico do site, oferecendo um sistema que permita visualizar a divisão de cada termo pela sua utilização nas várias áreas científicas suportadas ou pela utilização dos constituintes morfológicos presentes no sistema. Para terminar devem de serem implementados mais mecanismos de segurança, como por exemplo a criação de um administrador: que é responsável pela gestão da informação (é único que pode inserir, modificar ou remover informação do sistema); ou a utilização de contas de utilizadores com vários níveis de permissões na gestão da base de dados.

5. Referências

1. Microsoft. *.NET Framework Developer Center*. (2009) [Recuperado em 25 de Agosto de 2009]; de http://msdn.microsoft.com/en-us/netframework/default.aspx.
2. Microsoft. *Microsoft SQL Server 2005 Home*. (2009) [Recuperado em 18 de Setembro de 2009]; de http://www.microsoft.com/SQL/default.mspx.
3. Microsoft. *ASP .NET AJAX*. (2009) [Recuperado em 8 de Setembro de 2009]; de http://www.asp.net/ajax/.
4. Francisco Cortés Gabaudan, Elena Cid Ledesma, Mª Teresa Cid Ledesma, Mª Concepción Ledesma Martín, Miguel Marchena, Pedro Pedrero, Jesús Ureña Bracero. *Dicciomed.es. Diccionario médico-biológico, histórico y etimológico*. (2007) [Recuperado em 10 de Agosto de 2009]; de http://www.dicciomed.es.
5. ILTEC, Instituto de Linguística Teórica e Computacional. *MorDebe*. (2009) [Recuperado em 20 de Outubro de 2009]; de http://www.portaldalinguaportuguesa.org/.
6. *Houaiss - Dicionário da Lingua Portuguesa*. (2009) [Recuperado em 20 de Outubro de 2009]; de http://portal.doc.ua.pt/houaiss/.
7. Contente, Maria Madalena Dias Marques, *Terminocriativide, sinonímia e equivalência interlinguística em medicina*. 2008, Lisboa: Edições Colibri.
8. PHP. *PHP: Hypertext Preprocessor*. (2009) [Recuperado em 10 de Setembro de 2009]; de http://www.php.net.
9. Sun, Sun microsystems. *MySQL:: The world's most popular open source database*. (2009) [Recuperado em 10 de Setembro de 2009]; de http://www.mysql.com.
10. mono. *mono*. (2009) [Recuperado em 25 de Agosto de 2009]; de http://mono-project.com/.
11. Troelsen, Andrew, *Pro C# 2008 and the .NET 3.5 Platform, Fourth Edition*. Fourth Edition ed. 2007: Apress.
12. Microsoft. *The Official Microsoft ASP.NET Site*. (2009) [Recuperado em 15 de Setembro de 2009]; de http://www.asp.net.
13. Duthie, G. Andrew and Matthew MacDonald, *ASP.NET in a Nutshell*. 2003: O'Reilly.
14. Liberty, Jesse and Donald Xie, *Programming C# 3.0, Fifth Edition*. 2008: O'Reilly.
15. W3C, World Wide Web Consortium. *Web Services @ W3C*. (2009) [Recuperado em 31 de Agosto de 2009]; de http://www.w3.org/2002/ws.
16. W3C, World Wide Web Consortium. *HTTP - Hypertext Transfer Protocol*. (2009) [Recuperado em 31 de Agosto de 2009]; de http://www.w3.org/Protocols/.
17. W3C, World Wide Web Consortium. *Extensible Markup Language (XML)*. (2009) 22-05-2008 [Recuperado em 31 de Agosto de 2009]; de http://www.w3.org/XML.
18. W3C, World Wide Web Consortium. *XML Schema*. (2009) [Recuperado em 31 de Agosto de 2009]; de http://www.w3.org/XML/Schema.
19. W3C, World Wide Web Consortium. *SOAP Specifications*. (2009) [Recuperado em 31 de Agosto de 2009]; de http://www.w3.org/TR/soap.
20. W3C, World Wide Web Consortium. *Web Service Definition Language (WSDL)*. (2009) [Recuperado em 31 de Agosto de 2009]; de http://www.w3.org/TR/wsdl.
21. OASIS. *UDDI Version 3.0.2*. (2009) [Recuperado em 31 de Agosto de 2009]; de http://www.uddi.org/pubs/uddi_v3.htm.
22. ECMA. *Standard ECMA-357*. (2008) [Recuperado em 20 de Agosto de 2009]; de http://www.ecma-international.org/publications/standards/Ecma-357.htm.
23. Powers, Shelley, ***Learning JavaScript*** 2006.

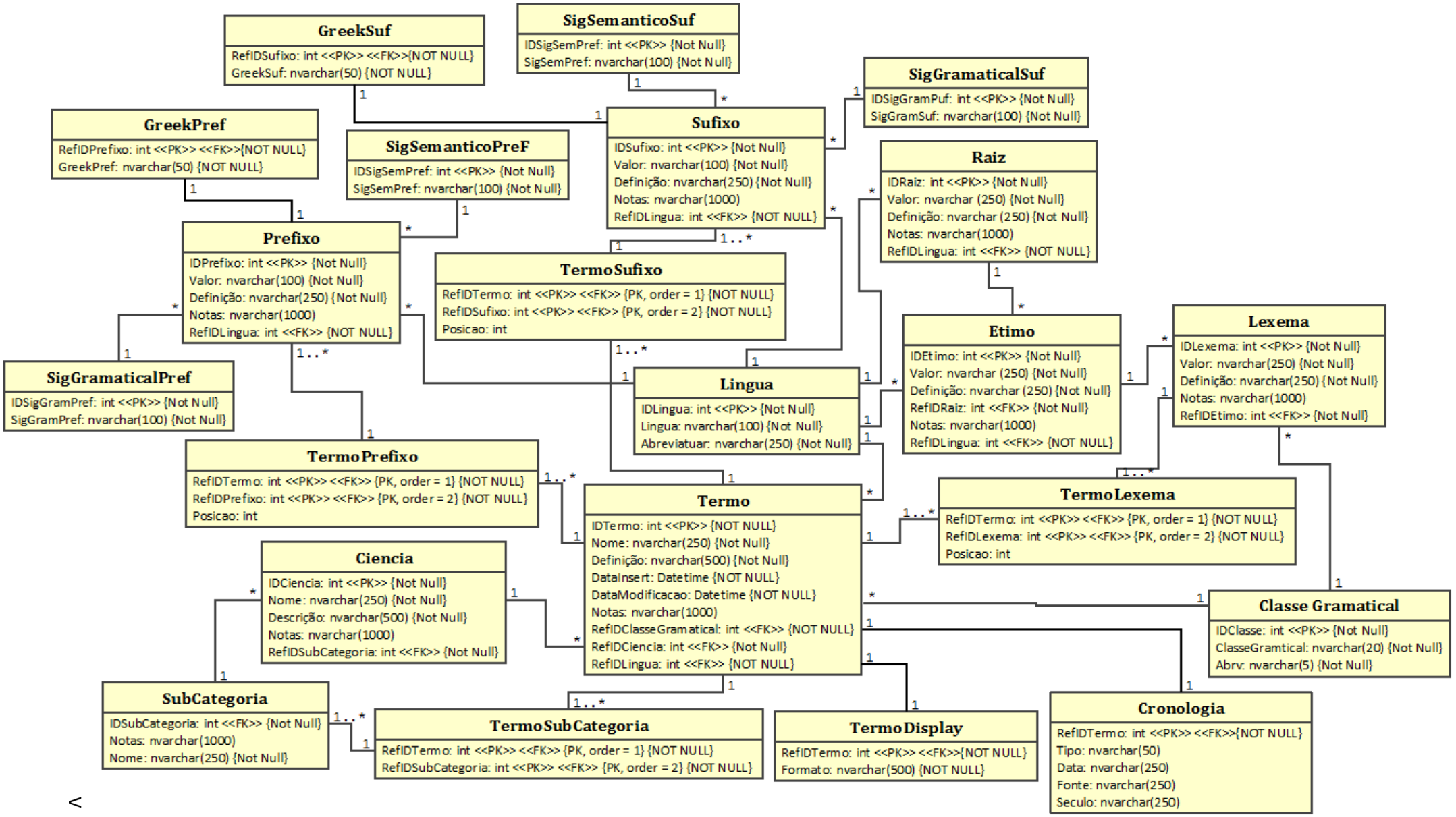
24. W3C, World Wide Web Consortium. *W3C Document Object Model*. (2009) [Recuperado em 31 de Agosto de 2009]; de http://www.w3.org/DOM.
25. W3C. *World Wide Web Consortium*. (2009) [Recuperado em 31 de Agosto de 2009]; de http://www.w3.org/.
26. W3C, World Wide Web Consortium. *Cascading Style Sheets*. (2009) [Recuperado em 31 de Agosto de 2009]; de http://www.w3.org/Style/CSS.
27. Wenz, Christian, *Programming ASP.NET AJAX*. 2007: O'Reilly.
28. Ben-Gan, Itzik, Dejan Sarka, and Roger Wolter, *Inside Microsoft SQL Server 2005: T-SQL Programming*. 2006: Microsoft Press.
29. Microsoft. *Visual Studio Developer Center*. (2009) [Recuperado em 10 de Setembro de 2009]; de http://msdn.microsoft.com/en-us/vstudio/default.aspx.
30. Sun, Sun microsystems. *The Source for Java Developers*. (2009) [Recuperado em 12 de Setembro de 2009]; de http://java.sun.com/.
31. Sun, Sun microsystems. *Java Platform, Enterprise Edition - Java EE*. (2009) [Recuperado em 25 de Setembro de 2009]; de http://java.sun.com/javaee/.
32. Sun, Sun microsystems. *JavaServer Pages*. (2009) [Recuperado em 25 de Setembro de 2009]; de http://java.sun.com/products/jsp/.
33. Apache, The Apache Software Foundation. *Apache Derby*. (2009) [Recuperado em 25 de Setembro de 2009]; de http://db.apache.org/derby/.
34. ORACLE. *Oracle 11g, Siebel, PeopleSoft | Oracle, Thw World's Largest Enterprise Software Company*. (2009) [Recuperado em 25 de Agosto de 2009]; de http://www.oracle.com/.
35. Apache, The Apache Software Foundation. *Apache Tomcat*. (2009) [Recuperado em 20 de Setembro de 2009]; de http://tomcat.apache.org/.
36. java.net. *GlassFish*. (2009) [Recuperado em 25 de Agosto de 2009]; de https://glassfish.dev.java.net/.
37. *Netbeans IDE*. (2009) [Recuperado em 25 de Agosto de 2009]; de http://www.netbeans.org/.
38. *Eclipse*. (2009) [Recuperado em 25 de Agosto de 2009]; de http://www.eclipse.org/.

Anexos

Índice de Anexos

Anexo A: Modelo Físico da Base de Dados

O modelo físico da base de dados encontra-se na página seguinte para melhor para uma melhor visualização.


GreekSuf
RefIDSufixo: int <<PK>> <<FK>>{NOT NULL}
GreekSuf: nvarchar(50) {NOT NULL}
SigSemanticoSuf
IDSigSemPref: int <<PK>> {Not Null}
SigSemPref: nvarchar(100) {Not Null}
SigGramaticalSuf
IDSigGramPuf: int <<PK>> {Not Null}
SigGramSuf: nvarchar(100) {Not Null}
GreekPref
RefIDPrefixo: int <<PK>> <<FK>>{NOT NULL}
GreekPref: nvarchar(50) {NOT NULL}
SigSemanticoPreF
IDSigSemPref: int <<PK>> {Not Null}
SigSemPref: nvarchar(100) {Not Null}
Sufixo
IDSufixo: int <<PK>> {Not Null}
Valor: nvarchar(100) {Not Null}
Definição: nvarchar(250) {Not Null}
Notas: nvarchar(1000)
RefIDLingua: int <<FK>> {NOT NULL}
Raiz
IDRaiz: int <<PK>> {Not Null}
Valor: nvarchar (250) {Not Null}
Definição: nvarchar (250) {Not Null}
Notas: nvarchar(1000)
RefIDLingua: int <<FK>> {NOT NULL}
Prefixo
IDPrefixo: int <<PK>> {Not Null}
Valor: nvarchar(100) {Not Null}
Definição: nvarchar(250) {Not Null}
Notas: nvarchar(1000)
RefIDLingua: int <<FK>> {NOT NULL}
TermoSufixo
RefIDTermo: int <<PK>> <<FK>> {PK, order = 1} {NOT NULL}
RefIDSufixo: int <<PK>> <<FK>> {PK, order = 2} {NOT NULL}
Posicao: int
Etimo
IDEtimo: int <<PK>> {Not Null}
Valor: nvarchar (250) {Not Null}
Definição: nvarchar (250) {Not Null}
RefIDRaiz: int <<FK>> {Not Null}
Notas: nvarchar(1000)
RefIDLingua: int <<FK>> {NOT NULL}
Lexema
IDLexema: int <<PK>> {Not Null}
Valor: nvarchar(250) {Not Null}
Definição: nvarchar(250) {Not Null}
Notas: nvarchar(1000)
RefIDEtimo: int <<FK>> {Not Null}
SigGramaticalPref
IDSigGramPref: int <<PK>> {Not Null}
SigGramPref: nvarchar(100) {Not Null}
Lingua
IDLingua: int <<PK>> {Not Null}
Lingua: nvarchar(100) {Not Null}
Abreviatuar: nvarchar(250) {Not Null}
TermoPrefixo
RefIDTermo: int <<PK>> <<FK>> {PK, order = 1} {NOT NULL}
RefIDPrefixo: int <<PK>> <<FK>> {PK, order = 2} {NOT NULL}
Posicao: int
TermoLexema
RefIDTermo: int <<PK>> <<FK>> {PK, order = 1} {NOT NULL}
RefIDLexema: int <<PK>> <<FK>> {PK, order = 2} {NOT NULL}
Posicao: int
Termo
IDTermo: int <<PK>> {NOT NULL}
Nome: nvarchar(250) {Not Null}
Definição: nvarchar(500) {Not Null}
DataInsert: Datetime {NOT NULL}
DataModificacao: Datetime {NOT NULL}
Notas: nvarchar(1000)
RefIDClasseGramatical: int <<FK>> {NOT NULL}
RefIDCiencia: int <<FK>> {Not Null}
RefIDLingua: int <<FK>> {NOT NULL}
Ciencia
IDCiencia: int <<PK>> {Not Null}
Nome: nvarchar(250) {Not Null}
Descrição: nvarchar(500) {Not Null}
Notas: nvarchar(1000)
RefIDSubCategoria: int <<FK>> {Not Null}
Classe Gramatical
IDClasse: int <<PK>> {Not Null}
ClasseGramtical: nvarchar(20) {Not Null}
Abrv: nvarchar(5) {Not Null}
SubCategoria
IDSubCategoria: int <<FK>> {Not Null}
Notas: nvarchar(1000)
Nome: nvarchar(250) {Not Null}
TermoSubCategoria
RefIDTermo: int <<PK>> <<FK>> {PK, order = 1} {NOT NULL}
RefIDSubCategoria: int <<PK>> <<FK>> {PK, order = 2} {NOT NULL}
TermoDisplay
RefIDTermo: int <<PK>> <<FK>>{NOT NULL}
Formato: nvarchar(500) {NOT NULL}
Cronologia
RefIDTermo: int <<PK>> <<FK>>{NOT NULL}
Tipo: nvarchar(50)
Data: nvarchar(250)
Fonte: nvarchar(250)
Seculo: nvarchar(250)

Anexo B: Implementação da Base de Dados

B1. Criação das tabelas

O diagrama do modelo físico da base de dados obtido anteriormente, torna mais fácil a criação do código de construção das tabelas, no *Microsoft SQL Server*. O código *T-SQL* utilizado na tabela Termo é o seguinte:

```
CREATE TABLE Termo
 (
  IDTermo INT IDENTITY
          PRIMARY KEY,
  Termo NVARCHAR(50) NOT NULL,
  Definicao NVARCHAR(2000) NOT NULL,
  Genero NVARCHAR(15) NOT NULL,
  Notas NVARCHAR(1000),
  DataInsert DATETIME NOT NULL,
  DataModificao DATETIME NOT NULL,
  RefIDClasseGramatical INT NOT NULL
                FOREIGN KEY REFERENCES ClasseGramatical ( IDClasse ),
  RefIDCiencia INT NOT NULL
            FOREIGN KEY REFERENCES Ciencia ( IDCiencia ),
  RefIDLingua INT NOT NULL
            FOREIGN KEY REFERENCES Lingua ( IDLingua )
 )
```

A informação que vai ser armazenada é fácil de identificar, sendo possível verificar qual será a chave primária da tabela em questão:

```
IDTermo INT IDENTITY PRIMARY KEY,
```

Para definir uma chave primária em *T-SQL*, é necessário utilizar a palavra-chave: *Identity Primary Key*.

A tabela termo apresenta ainda, duas chaves estrangeiras:

```
RefIDClasseGramatical INT NOT NULL
            FOREIGN KEY REFERENCES ClasseGramatical ( IDClasse ),
RefIDCiencia INT NOT NULL
            FOREIGN KEY REFERENCES Ciencia ( IDCiencia )
```

Uma chave estrangeira é construída através da palavra-chave do T-SQL: *Foreign Key Reference*. Esta palavra-chave é então seguida da tabela a referenciar, sendo indicada depois, dentro de parênteses curvos, a chave a utilizar.

Na tabela termo, as chaves estrangeiras referenciam as chaves primárias das tabelas ClasseGramatical e Ciência, ou seja, duas tabelas relacionadas com a tabela Termo. Então o código *T-SQL* dessas tabelas é:

```
CREATE TABLE Ciencia
   (
    IDCiencia INT IDENTITY
              PRIMARY KEY,
    Ciencia NVARCHAR(50) NOT NULL,
    Descricao NVARCHAR(500),
    Notas NVARCHAR(1000)
   )
```

```
CREATE TABLE ClasseGramatical
   (
    IDClasse INT IDENTITY
             PRIMARY KEY,
    Classe NVARCHAR(20) NOT NULL,
    Abv NVARCHAR(5) NOT NULL
   )
```

Uma outra tabela relacionada com a tabela termo, é a tabela Cronologia, sendo este relacionamento feito de uma forma diferente:

```
CREATE TABLE Cronologia
   (
    RefIDTermo INT NOT NULL
               PRIMARY KEY
               FOREIGN KEY REFERENCES Termo ( IDTermo ),
    Tipo NVARCHAR(50) NOT NULL,
    Data NVARCHAR(250) NOT NULL,
    DataSource NVARCHAR(250) NOT NULL,
    Seculo NVARCHAR(250) NOT NULL,
   )
```

A chave primária desta tabela é composta por uma chave estrangeira que referencia a tabela termo. Para criar uma chave primária deste género devem ser adicionadas as palavras-chave de definição de chave primária com a chave estrangeira. Em primeiro lugar, são utilizadas palavras-chave para definir a chave primária, seguidas das palavras-chave para definir a chave estrangeira e da tabela a referenciar.

Como mencionado anteriormente, foi necessário criar uma nova tabela para expressar a relação existente entre a informação de um Termo e de um Lexema:

```
CREATE TABLE TermoLexema
   (
    RefIDTermo INT NOT NULL
            FOREIGN KEY REFERENCES Termo ( IDTermo ),
    RefIDLexema INT NOT NULL
             FOREIGN KEY REFERENCES Lexema ( IDLexema ),
    Posicao INT NOT NULL,
    CONSTRAINT PK_TERMOLEXEMA primary key ( RefIDTermo,
RefIDLexema ),
   )
```

Esta tabela irá apresentar como chave primária, uma combinação de duas chaves estrangeiras, que representam a relação existente entre as duas tabelas, Termo e Lexema.

```
    RefIDTermo INT NOT NULL
            FOREIGN KEY REFERENCES Termo ( IDTermo ),
    RefIDLexema INT NOT NULL
             FOREIGN KEY REFERENCES Lexema ( IDLexema ),
```

Devido a esta combinação, a chave primária é considerada uma chave primária composta. A definição dessa chave para tabelas deste género é feita de forma diferente, através da utilização de uma restrição do tipo *primary key*, indicando quais as colunas que irão constituir a chave primária.

```
CONSTRAINT PK_TERMOLEXEMA primary key ( RefIDTermo, RefIDLexema ),
```

A restrição é criada através da palavra-chave *constrain*, seguida do nome e do tipo de restrição, neste caso *primary key*, e por fim, entre parênteses curvos, das duas chaves estrangeiras. Esta restrição, para além de ser a chave primária da tabela, vai impedir que um mesmo termo apresente o mesmo prefixo em posições diferentes, contribuindo assim para a integridade dos dados.

Como verificado durante a construção do modelo físico da base de dados, irão surgir mais tabelas deste género. A criação do código das mesmas é idêntica ao apresentado anteriormente. As tabelas são a TermoSufixo, TermoPrefixo e TermoSubCat. O código de implementação destas tabelas e das restantes é apresentado a seguir:

```
CREATE TABLE Lingua
 (
  IDLingua INT IDENTITY
           PRIMARY KEY,
  Lingua NVARCHAR(50) NOT NULL,
  Abv NVARCHAR(5) NOT NULL
 )

CREATE TABLE SubCategoriaCien
 (
  IDSubCat INT IDENTITY
           PRIMARY KEY,
  SubCategoriaCien NVARCHAR(50) NOT NULL,
  Descricao NVARCHAR(500),
  Notas NVARCHAR(1000),
  RefIDCiencia INT NOT NULL
               FOREIGN KEY REFERENCES Ciencia ( IDCiencia )
 )

CREATE TABLE TermoDisplay
 (
  RefIDTermo INT NOT NULL
             PRIMARY KEY
             FOREIGN KEY REFERENCES Termo ( IdTermo ),
  Formato NVARCHAR(500) NOT NULL
 )

CREATE TABLE TermoSubCategoria
 (
  RefIDTermo INT NOT NULL
             FOREIGN KEY REFERENCES Termo ( IDTermo ),
  REfIDSubCat INT NOT NULL
              FOREIGN KEY REFERENCES SubCategoriaCien ( IDSubCat )
  CONSTRAINT PK_TermoSubCat primary key ( RefIDTermo, RefIDSubCat )
 )

CREATE TABLE SigSemanticoPref
 (
  IDSigSemPref INT IDENTITY
               PRIMARY KEY,
  SigSemPref NVARCHAR(50) NOT NULL
 )

CREATE TABLE SigGramaticalPref
 (
  IDSigGramPref INT IDENTITY
                PRIMARY KEY,
  SigGramPref NVARCHAR(50) NOT NULL
 )
```

```
CREATE TABLE Prefixo
 (
  IDPrefixo INT IDENTITY
            PRIMARY KEY,
  Prefixo NVARCHAR(50) NOT NULL,
  Definicao NVARCHAR(500) NOT NULL,
  Notas NVARCHAR(1000),
  RefIDSigSemPref INT NOT NULL
                  FOREIGN KEY REFERENCES SigSemanticoPref ( IDSigSemPref ),
  RefIDSigGramPref INT NOT NULL
                   FOREIGN KEY REFERENCES SigGramaticalPref ( IDSigGramPref ),
  RefIDLingua INT NOT NULL
              FOREIGN KEY REFERENCES Lingua ( IDLingua )
 )
CREATE TABLE GreekPref
 (
  RefIDPrefixo INT NOT NULL
               PRIMARY KEY
               FOREIGN KEY REFERENCES Prefixo ( IDPrefixo ),
  GreekPref NVARCHAR(50) NOT NULL
 )
CREATE TABLE TermoPrefixo
 (
  RefIDTermo INT FOREIGN KEY REFERENCES Termo ( IDTermo ),
  RefIDPrefixo INT FOREIGN KEY REFERENCES Prefixo ( IDPrefixo ),
  Posicao INT NOT NULL
  CONSTRAINT PK_Prefixo primary key ( RefIDTermo, RefIDPrefixo ),
 )
CREATE TABLE SigSemanticoSuf
 (
  IDSigSemSuf INT IDENTITY
              PRIMARY KEY,
  SigSemSuf NVARCHAR(50) NOT NULL
 )
CREATE TABLE SigGramaticalSuf
 (
  IDSigGramSuf INT IDENTITY
               PRIMARY KEY,
  SigGramSuf NVARCHAR(50) NOT NULL
 )

CREATE TABLE Sufixo
 (
  IDSufixo INT IDENTITY
           PRIMARY KEY,
  Sufixo NVARCHAR(50) NOT NULL,
  Definicao NVARCHAR(500) NOT NULL,
  Notas NVARCHAR(1000),
  RefIDSigSemSuf INT NOT NULL
                 FOREIGN KEY REFERENCES SigSemanticoSuf ( IDSigSemSuf ),
  RefIDSigGramSuf INT NOT NULL
                  FOREIGN KEY REFERENCES SigGramaticalSuf ( IDSigGramSuf ),
  RefIDLingua INT NOT NULL
              FOREIGN KEY REFERENCES Lingua ( IDLingua )
 )


CREATE TABLE GreekSuf
```

```
 (
  RefIDSufixo INT NOT NULL
              PRIMARY KEY
              FOREIGN KEY REFERENCES Sufixo ( IDSufixo ),
  GreekSuf NVARCHAR(50)
 )
CREATE TABLE TermoSufixo
 (
  RefIDTermo INT NOT NULL
             FOREIGN KEY REFERENCES Termo ( IDTERMO ),
  RefIDSufixo INT NOT NULL
              FOREIGN KEY REFERENCES Sufixo ( IDSufixo ),
  Posicao INT NOT NULL
  CONSTRAINT PK_TermoStat primary key ( RefIDTermo, RefIDSufixo ),

 )
CREATE TABLE Raiz
 (
  IDRaiz INT IDENTITY
         PRIMARY KEY,
  Raiz NVARCHAR(50) NOT NULL,
  Definicao NVARCHAR(500) NOT NULL,
  Notas NVARCHAR(1000),
  RefIDLingua INT NOT NULL
              FOREIGN KEY REFERENCES Lingua ( IDLingua )
 )

CREATE TABLE Etimo
 (
  IDEtimo INT IDENTITY
          PRIMARY KEY,
  Etimo NVARCHAR(50) NOT NULL,
  Definicao NVARCHAR(500) NOT NULL,
  Notas NVARCHAR(1000),
  RefIDRaiz INT NOT NULL
            FOREIGN KEY REFERENCES Raiz ( IDRaiz ),
  RefIDLingua INT NOT NULL
              FOREIGN KEY REFERENCES Lingua ( IDLingua )
 )
CREATE TABLE Lexema
 (
  IDLexema INT IDENTITY
           PRIMARY KEY,
  Lexema NVARCHAR(50) NOT NULL,
  Conceito NVARCHAR(50) NOT NULL,
  Notas NVARCHAR(1000),
  RefIDEtimo INT NOT NULL
             FOREIGN KEY REFERENCES Etimo ( IDEtimo ),
  RefIDClasse INT NOT NULL
              FOREIGN KEY REFERENCES ClasseGramatical ( IDClasse )
 )

create TABLE TermoLexema
 (
  RefIDTermo INT NOT NULL
             FOREIGN KEY REFERENCES Termo ( IDTermo ),
  RefIDLexema INT NOT NULL
              FOREIGN KEY REFERENCES Lexema ( IDLexema ),
```

```
   Posicao INT NOT NULL,
   CONSTRAINT PK_TERMOLEXEMA primary key ( RefIDTermo, RefIDLexema ),
  )

--ESTATISTICAS
CREATE TABLE Data
  (
   IDData INT IDENTITY
          PRIMARY KEY,
   Mes NVARCHAR(20) NOT NULL,
   Ano INT NOT null
  )

CREATE TABLE Stat
  (
   RefIDTermo INT NOT NULL
            FOREIGN KEY REFERENCES Termo ( IDTermo ),
   RefIDData INT NOT NULL
            FOREIGN KEY REFERENCES Data ( IDData ),
   NPesquisas INT NOT NULL
   CONSTRAINT PK_Stat primary key ( RefIDTermo, RefIDData )
  )
```

B2. Criação de índices

Em algumas tabelas não é necessária redundância de valores, isto é, a mesma informação não precisa de ser repetida várias vezes. Aliás, a redundância da informação num SGBD deve ser evitada para impedir o aumento desnecessário de uma base de dados. De forma a evitar a redundância é utilizado, em algumas tabelas, um índice do tipo *unique*, que impede a repetição de valores numa coluna específica. Foram criados índices deste tipo para as seguintes tabelas:

Língua:

```
CREATE UNIQUE INDEX IX_LINGUA ON Lingua ( Lingua )
```

Ciência:

```
CREATE UNIQUE INDEX IX_CIENCIA ON Ciencia ( Ciencia )
```

Subdomínio científico:

```
CREATE UNIQUE INDEX IX_SUBCATEGORIACIEN ON SubCategoriaCien ( SubCategoriaCien )
```

ClasseGramatical:

```
CREATE UNIQUE INDEX IX_CATGRAMATICAL ON ClasseGramatical ( Classe )
```

TermoDisplay:

```
CREATE UNIQUE INDEX IX_CATGRAMATICAL ON ClasseGramatical ( Classe )
```

Como verificado anteriormente, um índice único é construído, utilizando as palavras-chave *unique índex* e seguido de um nome. De seguida é especificada a tabela e a coluna em que o índice vai ser aplicado, entre parênteses curvos.

B3. *Stored Procedures*

De forma a facilitar algumas operações realizadas pela aplicação Web desenvolvida, foram criados dois tipos de *stored procedures*: um que verifica a existência de informação na base de dados e outro que realiza a inserção de informação na base de dados.

A seguir é apresentado o *stored procedure* que verifica a existência de um dado termo na base de dados.

```
-- Checks if Termo exist
-- If exists returns the Termo ID
-- If not exists returns -1
GO
CREATE PROCEDURE CheckTermo
 @checkTermo AS NVARCHAR(50),
 @flag AS INT OUTPUT
AS
 BEGIN
  SET @flag=-1 -- Termo not exists
  IF EXISTS ( SELECT  Termo.IDTermo
          FROM    Termo
          WHERE   Termo.Termo=@checkTermo )
   BEGIN
    SET @flag=( SELECT IDTermo FROM Termo WHERE Termo = @checkTermo )
   END
 END
GO
```

Este *stored procedure* é essencialmente uma simples *query* de *select*, que aceita um parâmetro de entrada, correspondente ao nome de um termo, de forma a verificar se já foi armazenado na base de dados. O *stored procedure* pode devolver dois valores: em caso de não ocorrência do termo na base de dados devolve menos um; em caso de ocorrência devolve o identificador único do termo em questão. O retorno deste número será útil no funcionamento da aplicação Web desenvolvida.

Foram criados mais *stored procedures* para verificar a existência de outros tipos de informação como: a língua, o lexema, o étimo, a raiz, o prefixo, o significado gramatical ou semântico de um prefixo, o sufixo e o significado gramatical ou semântico de um sufixo, sendo a sua construção idêntica, alterando naturalmente a tabela onde se realiza a consulta.

Os *stored procedures* do segundo tipo são um pouco mais complexos que os anteriores. Como são responsáveis pela inserção de informação na base de dados, apresentam vários parâmetros de entrada que devem ser preenchidos de forma correcta. A validação e o preenchimento desses parâmetros estão a cargo da aplicação Web.

Um exemplo de um *stored procedure* de inserção é o seguinte:

```
GO

CREATE PROCEDURE InsertPrefixo
  @prefixoIn NVARCHAR(50),
  @definicaoIn NVARCHAR(500),
  @notasIn NVARCHAR(50),
  @reflDSsemIn INT,
  @reflDSgramIn INT,
  @reflDLinguaIn INT
AS
  BEGIN TRANSACTION
  INSERT  INTO Prefixo ( Prefixo, Definicao, Notas, ReflDSigSemPref, ReflDSigGramPref,
ReflDLingua )
  VALUES  ( /* Prefixo - nvarchar(50) */ @prefixoIn,
            /* Definicao - nvarchar(500) */ @definicaoIn,
            /* Notas - nvarchar(1000) */ @notasIn,
            /* ReflDSigSemPref - int */ @reflDSsemIn,
            /* ReflDSigGramPref - int */ @reflDSgramIn,
            /* ReflDLingua - int */ @reflDLinguaIn )

  IF @@ERROR <> 0
    BEGIN
      ROLLBACK TRANSACTION
      RETURN 1
    END
  COMMIT TRANSACTION
  RETURN @@IDENTITY
```

Este *stored procedure* é responsável pela inserção dos dados na tabela Prefixo. Para os dados que são armazenados nas tabelas relacionadas com a tabela Prefixo, o seu identificador único é passado para cada um deles. A inserção ou obtenção, bem como a sua validação são da responsabilidade da aplicação Web desenvolvida.

Foram construídos mais *stored procedures* deste género, que apresentam a mesma implementação. Foram criados para a inserção de informação nas tabelas Lexema, Étimo, Raiz, Sufixo, Ciência, SubCatCien, CaracTermo e ClasseGramatical.

O *stored procedure* mais complexo é o responsável pela inserção da informação de um termo. Para além disso, é também responsável pela inserção da cronologia e associação do termo aos seus respectivos subdomínios científicos e constituintes morfológicos, ou seja, insere a informação nas tabelas Cronologia, TermoSubdominio, TermoLexema, TermoPrefixo e TermoSufixo. Como este *stored procedure* é extenso é melhor analisá-lo por partes.

O *stored procedure* denomina-se *InsertTermo* e apresenta um conjunto amplo de dados de entrada que são validados e enviados pela aplicação Web:

```
-- Insert Termo
GO
Create PROCEDURE InsertTermo
  @termoIn NVARCHAR(50), @definicaoIn NVARCHAR(2000), @notasIn NVARCHAR(1000),
  @genero NVARCHAR(15), @refIDClasseIn INT, @refIDCienciaIn INT, @refIDSubdominio INT,
  @refIDLingua INT, @refIDPrefixo INT, @posicaoPrefixo INT, @refIDSufixo INT, @posicaoSufixo INT,
  @refIDLexema INT, @posicaoLexema INT, @carac INT, @tipo NVARCHAR(50), @data NVARCHAR(250),
  @fonte NVARCHAR(250),  @Seculo NVARCHAR(250)
AS
  DECLARE @idtermo INT
 SET @idtermo = -1
```

A inserção dos dados respectivos à tabela termo é feita da seguinte forma:

```
BEGIN TRANSACTION
  -- verificar se o termo existe:
  IF NOT EXISTS ( SELECT  Termo.IDTermo FROM    Termo   WHERE   Termo.Termo = @termoIn )
    BEGIN
      INSERT  INTO dbo.Termo ( Termo, Definicao, Notas, Genero, DataInsert, DataModificao,
RefIDClasseGramatical, RefIDCiencia, RefIDLingua )
      VALUES  ( /* Termo - nvarchar(50) */ @termoIn,
                /* Definicao - nvarchar(2000) */ @definicaoIn,
                /* Notas - nvarchar(1000) */ @notasIn,
                /* Genero - nvarchar(15)*/ @genero,
                /* DataInsert - datetime */ GETDATE(),
                /* DataModificao - datetime */ GETDATE(),
                /* RefIDCatGramatical - int */ @refIDClasseIn,
                /* RefIDCiencia - int */ @refIDCienciaIn,
                /* RefIDLingua - int */ @refIDLingua )
      SET @idtermo = @@IDENTITY
    END
  ELSE
    BEGIN
      SET @idtermo = (SELECT  Termo.IDTermo  FROM    Termo WHERE   Termo.Termo = @termoIn)
    END
  IF @@ERROR <> 0
    BEGIN
      ROLLBACK TRANSACTION
      RETURN -1
    END
```

De notar presença das palavras-chave *BEGIN TRANSACTION.* Com a utilização destas palavras-chave todas as operações definidas são tratadas como uma única transacção. Isto

possibilita uma protecção para a integridade dos dados na base de dados, porque caso ocorra algum erro em alguma operação de inserção presente na transacção, esta termina e as alterações, realizadas nos dados até essa altura, são canceladas, fazendo a base de dados voltar ao estado em que estava anteriormente. Este mecanismo chama-se *rollback*.

A detecção de um erro é feita através desta parte de código:

```
IF @@ERROR <> 0
  BEGIN
            ROLLBACK TRANSACTION
            RETURN -1
  END
```

O *SQL Server* armazena numa variável de sistema *@@Error*, o código do erro ocorrido. Caso tenha ocorrido um erro, o valor desta variável será diferente de zero e a transacção cancelada (*rollback transaction*), terminando a execução do *stored procedure*.

Depois da inserção do termo ter sido realizada com sucesso, é realizada a inserção dos restantes dados. O modo de implementação para essas operações é análogo ao exibido anteriormente. Aqui está um exemplo do código de inserção de um Lexema:

```
IF @refIDPrefixo !=-1
   BEGIN
  --TERMOPREFIXO
  --verificar se o prefixo já está associado!!
     IF NOT EXISTS (SELECT posicao FROM dbo.TermoPrefixo WHERE
RefIDTermo = @idtermo AND RefIDPrefixo = @refIDPrefixo)
       BEGIN

         BEGIN
          INSERT  INTO dbo.TermoPrefixo
                ( RefIDTermo, RefIDPrefixo, Posicao )
          VALUES  ( /* RefIDTermo - int */ @idtermo,
                    /* RefIDPrefixo - int */ @refIDPrefixo,
                    /* Posicao - int */ @posicaoPrefixo )
         END
       END
     IF @@ERROR <> 0
       BEGIN
         ROLLBACK TRANSACTION
         PRINT 'ROLLBACK -2'
         RETURN -2
       END
   END
```

B4. *Views*

Como a informação se encontra dispersa num conjunto de tabelas, para facilitar a sua recolha e utilização na visualização da mesma por parte da aplicação Web, foram criadas *views*. A informação foi dividida em quatro grupos relevantes a um termo, a um lexema, a um prefixo e a um sufixo. Como já foi mencionado anteriormente, as *views* podem ser constituídas por informação presente em tabelas distintas e apresentam um funcionamento igual a uma tabela real.

A *view* que recolhe a informação relativa a um termo é a seguinte:

```
Create VIEW ViewTermo
AS  SELECT  t.IDTermo,
        t.Termo,
        t.Definicao,
        L.Lingua,
        ct.Data,
        ct.DataSource,
        ct.Seculo,
        ct.Tipo,
        c.IDCiencia,
        c.Ciencia,
        sc.IDSubCat,
        sc.SubCategoriaCien,
        t.genero,
        t.notas
   FROM    Termo T
        INNER JOIN Lingua L ON T.RefIDLingua = L.IDLingua
        INNER JOIN Ciencia C ON t.RefIDCiencia = c.IDCiencia
        LEFT JOIN Cronologia CT ON T.IDTermo = CT.RefIDTermo
        LEFT JOIN TermoSubCategoria TS ON TS.RefIDTermo = T.IDTermo
        left JOIN SubCategoriaCien SC ON TS.REfIDSubCat = SC.IDSubCat
```

A criação de uma *View* é feita pela utilização das palavras-chave do *T-SQL create view*, sendo essencialmente uma *query* de *select* que recolhe toda a informação desejada. Essa recolha de informação é feita através de *inner joins*, que combinam a informação existente nas tabelas. É de notar a utilização de *left joins* com as tabelas Cronologia, TermoSubCatategoria e SubCaegoriatCien, que permite a recolha de todos os termos que apresentem ou não dados nessas tabelas.

Estas *Views* são depois utilizadas na aplicação Web para visualização completa da informação.

As restantes *Views* seguem a mesma implementação apresentada como é possível verificar a seguir:

***View* para os prefixos:**

```
Create VIEW ViewPrefixo
as
  Select  P.IDPrefixo, P.Prefixo, P.Definicao, P.Notas, G.GreekPref,
        SG.SigGramPref, SM.SigSemPref, L.Lingua, L.Abv
  from    Prefixo P
          inner join SigGramaticalPref SG on SG.IDSigGramPref = P.RefIDSigGramPref
          inner join SigSemanticoPref SM on SM.IDSigSemPref = P.RefIDSigSemPref
          inner join Lingua L on L.IDLingua = P.RefIDLingua
          left join dbo.GreekPref G
        ON G.RefIDPrefixo = P.IDPrefixo
```

***View* para os lexemas:**

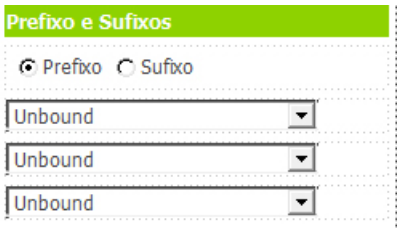
```
create VIEW ViewLexema
AS
  SELECT  L.IDLexema, L.Lexema, L.Conceito, L.Notas, E.Etimo,
          LG.Lingua AS LinguaE, LG.Abv AS LinguaAbvE, R.Raiz, R.Definicao,
          LGR.Lingua, CG.Classe, R.IDRaiz
   FROM    Lexema L
            INNER JOIN Etimo E ON L.RefIDEtimo = E.IDEtimo
            INNER JOIN Lingua LG ON E.RefIDLingua = LG.IDLingua
            INNER JOIN Raiz R ON R.IDRaiz = E.RefIDRaiz
            INNER JOIN dbo.Lingua LGR ON R.RefIDLingua = LGR.IDLingua
            INNER JOIN dbo.ClasseGramatical CG
          ON CG.IDClasse = L.RefIDClasse
```

***View* para os sufixos:**

```
create VIEW ViewSufixo
as
  Select  S.IDSufixo, S.Sufixo, S.Definicao, S.Notas, G.GreekSuf,
        SG.SigGramSuf, SM.SigSemSuf, L.Lingua, L.abv
  from    Sufixo S
          inner join SigGramaticalSuf SG on SG.IDSigGramSuf = S.RefIDSigGramSuf
          inner join SigSemanticoSuf SM on SM.IDSigSemSuf = S.RefIDSigSemSuf
          inner join Lingua L on L.IDLingua = S.RefIDLingua
          left join dbo.GreekSuf G
        ON G.RefIDSufixo = S.IDSufixo
```

Anexo C: *Master Page.aspx*

C1. Exemplo da definição de controlos *aspx*



O código seguinte exemplifica a construção dos controlos exibidos na figura anterior.

```
            <td class="myheader">
              <strong>Prefixo e Sufixos</strong>
            </td>
          </tr>
          <tr>
            <td>
              <asp:RadioButtonList ID="RBPrefSuf" runat="server" RepeatDirection="Horizontal"
CssClass="labeltxt"
                AutoPostBack="True" OnSelectedIndexChanged="RBPrefSuf_SelectedIndexChanged">
                <asp:ListItem Selected="True">Prefixo</asp:ListItem>
                <asp:ListItem>Sufixo</asp:ListItem>
              </asp:RadioButtonList>
            </td>
          </tr>
          <tr>
            <td>
              <asp:DropDownList ID="DDPrefSuf" runat="server" CssClass="labeltxt" Width="80%"
OnSelectedIndexChanged="DDPrefSuf_SelectedIndexChanged"
                AutoPostBack="true">
              </asp:DropDownList>
            </td>
          </tr>
          <tr>
            <td>
              <asp:DropDownList ID="DDSigSem" runat="server" CssClass="labeltxt" Width="80%"
OnSelectedIndexChanged="DDSigSem_SelectedIndexChanged"
                AutoPostBack="true">
              </asp:DropDownList>
            </td>
          </tr>
          <tr>
            <td>
              <asp:DropDownList ID="DDSigGram" runat="server" CssClass="labeltxt" Width="80%"
OnSelectedIndexChanged="DDSigGram_SelectedIndexChanged"
                AutoPostBack="true">
              </asp:DropDownList>
            </td>
          </tr>
```

C2. Implementação do mapa do site e do menu

O mapa do site é definido no ficheiro XML denominado de *web.sitemap*. A localização desse ficheiro é indicada no ficheiro de configuração geral da aplicação, denominado *web.config* e que está colocado na raiz da aplicação. O código de definição do mapa do site do DiCiTer é apresentado a seguir:

```
<siteMap defaultProvider="smProvider1">
   <providers>
    <add name="smProvider1" type="System.Web.XmlSiteMapProvider" siteMapFile="~/Web.sitemap"/>
   </providers>
 </siteMap>
```

De realçar que é possível a definição de mapas distintos do site que podem ser utilizados por vários menus. O conteúdo do ficheiro XML *web.sitemap*, definido em cima é o seguinte:

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
  <siteMapNode url="~/" title="Home"  description="Home page">
    <siteMapNode url="" title="Visualizar"  description="Visualizar">
      <siteMapNode url="~/Selects/ViewCiencia.aspx" title="Ciencia" description="Visualizar Ciencia"/>
      <siteMapNode url="~/Selects/ViewLexema.aspx" title="Lexema" description="Visualizar Lexema"/>
      <siteMapNode url="~/Selects/ViewPrefixo.aspx" title="Prefixo" description="Visualizar Prefixo"/>
      <siteMapNode url="~/Selects/ViewSufixo.aspx" title="Sufixo" description="Visualizar Prefixo"/>
      <siteMapNode url="~/Selects/ViewTermo.aspx" title="Termo" description="Visualizar Termo"/>
    </siteMapNode>
    <siteMapNode url="" title="Inserir"  description="Inserção de conteúdos">
      <siteMapNode url="~/Inserts/InsertTermo.aspx" title="Termo" description="Inserir Termo"/>
      <siteMapNode url="~/Inserts/InsertCiencia.aspx" title="Ciencia"  description="Inserir Ciencia" />
      <siteMapNode url="~/Inserts/InsertSubCatCien.aspx" title="Subdomínio Científico"  description="Inserir
SubDominio Científico" />
      <siteMapNode url="~/Inserts/InsertSufixo.aspx" title="Sufixo" description="Inserir Sufixo"/>
      <siteMapNode url="~/Inserts/InsertPrefixo.aspx" title="Prefixo" description="Inserir Prefixo"/>
      <siteMapNode url="~/Inserts/InsertLexema.aspx" title="Lexema" description="Inserir Lexema"/>
    </siteMapNode>
  </siteMapNode>
</siteMap>
```

As configurações do ficheiro apresentado acima são utilizadas para preencher um controlo, denominado de SiteMapDataSource, que irá conter o mapa do site para futura utilização. A definição desse controlo é a seguinte:

```
<asp:SiteMapDataSource ID="SiteMapDataSource1" SiteMapProvider="smProvider1" runat="server" />
 <div id="container">
```

Depois disto tudo, é possível definir o menu, utilizando o controlo anterior como a fonte dos dados, tal como é possível verificar no código seguinte.

```
<td align="center" class="TopImg" colspan="3">
        <asp:Menu ID="Menu1" runat="server" DataSourceID="SiteMapDataSource1"
DynamicEnableDefaultPopOutImage="False"
          StaticEnableDefaultPopOutImage="False" Font-Names="Helvetica" Font-Size="12px"
          Orientation="Horizontal" StaticDisplayLevels="2" CssClass="IE8Fix" SkinID="mainmenu">
        </asp:Menu>
      </td>
```

O aspecto que o menu apresenta é realizado por estilos CSS. Mas antes de abordar a configuração do aspecto, convém realçar que na versão utilizada do controlo, o menu apresentava um erro de funcionamento no Microsoft Internet Explorer 8. Esse erro pode ser corrigido através de um estilo CSS *IE8Fix.* Em relação ao aspecto do menu, ele é definido através de um ficheiro de *skin*, denominado de Skins.skin e colocado na pasta App_Themes/Skin/. O código de esse ficheiro é o seguinte:

```
<asp:Menu SkinID="mainmenu" Orientation="Horizontal" runat="server" StaticDisplayLevels="2">
            <%-- Para os elementos que estão sempre visíveis --%>
            <StaticMenuStyle  CssClass="IE8Fix" BackColor="Transparent"  />
            <StaticMenuItemStyle CssClass="itemStatic" ItemSpacing="0" />
            <StaticHoverStyle CssClass="hover" />
            <StaticSelectedStyle CssClass="selected" />
            <%-- Para os elementos que vão aparecer quando o rato passa por cima do menu --%>
            <DynamicMenuStyle CssClass="IE8Fix"  BackColor="Transparent" />
            <DynamicMenuItemStyle CssClass="item" ItemSpacing="0" />
            <DynamicHoverStyle CssClass="hover" />
            <DynamicSelectedStyle CssClass="selected" />
</asp:Menu>
```

Os estilos CSS correspondentes são definidos num ficheiro CSS denominado de menu.css.

```css
/* Fix menu in internet explorer 8*/
.IE8Fix
{
  z-index: 100;
}

.item
{
  font-family: Tahoma,Helvetica,Helv;
  font-size: 12px;
  font-weight: bold;
  color: #444444;
  padding: 2px 2px 2px 2px;
  border-width: 1px 0 0 1px;
  border-style: solid;
  border-color: #FFFFFF;
  background-color: #8ED300; /* COR DE FUNDO */
  color: white; /*COR DO TEXTO*/
  text-align: left;
  width: 150px;
  z-index: 100;
}
```

```css
.itemStatic
{
  font-family: Tahoma,Helvetica,Helv;
  font-size: 12px;
  font-weight: bold;
  color: #444444;
  padding: 2px 2px 2px 2px;
  color: white; /*COR DO TEXTO*/
  text-align: center;
  z-index: 100;
}

.selected
{
  /*background-color:#FF9900; */
  background-color: #8ED300;
  color: #FFFFFF;
  font-weight: bolder;
  z-index: 100;
}
.hover
{
  color: black;
  background-color: #DADADA;
  z-index: 100;
}
```

Anexo D: Default.aspx

D1. Código da função C# que busca o último termo inserido

```
/// <summary>
  /// Obter o ID do  último termo armazenado na base de dados e exibi-lo
  /// </summary>
  private void GetLastTermo()
  {
    //Criar a ligação ao SGDB
    SqlConnection con = new SqlConnection(connectionString);
    //String que contém o T-SQL a executar, que obtém o id do último termo existente na base de dados
    string sql = "select Top 1 idtermo from Termo ORDER by idtermo desc";
    string id;
    try
    {
      //Executar a operação T-SQL
      SqlCommand cmd = new SqlCommand(sql, con);
      //Abrir a ligação
      con.Open();
      //Obter o resultado da operação T-SQL
      id = cmd.ExecuteScalar().ToString();
      if (!string.IsNullOrEmpty(id))
        //Redireccionar para o módulo de visualização de termos passando o id na querystring
        Response.Redirect("~/Selects/ViewTermo.aspx?id=" + id);
    }
    catch (SqlException err)
    {
      //Capturar erros provenientes do servidor de base de dados
      LBSqlError.Text = "SQL Error (GetLastTermo): " + err.Message + " -- " + err.LineNumber;
    }
    catch (Exception err)
    {
      //Capturar outros tipo de erro
      LBSqlError.Text = "Generic Error (GetLastTermo): " + err.Message;
    }
    finally
    {
      //Fechar a ligação
      con.Close();
    }
  }
```

D2. Código da função C# que analisa a *querystring*

```
/// <summary>
 /// Realizar a análise da querystring
 /// </summary>
 /// <param name="querystring"> querystring a analisar
 /// </param>
 private void ParseQueryString(string querystring)
 {
  string search = null;

  //Separa a query string pelo caracter igual
  string[] querySplit = querystring.Split('=');
  search = querySplit[1];

  //Substituir todos caracteres que não se encontram na expressão regular
  //Todos os caracteres menos os alfanuméricos (\n), os espaços (\s) e os - _ *
  search = Regex.Replace(search, @"[^\w\s-_*]", "");
  //Substituir o caracter * (wildcard para pesquisa) pelo wildcard do SQL Server (%)
  search = search.Replace('*', '%');

  //Remover espaços extra, transformandos-os em exactamente um
  RegexOptions Options = RegexOptions.IgnorePatternWhitespace | RegexOptions.Singleline;
  Regex regex = new Regex(@"\s{2,}", Options);
  search = regex.Replace(search.Trim(), " ");

  string sem = null;
  string gram = null;
  string ciencia = null;
  //Realizar a análise e invocar as funções respectivas
  switch (querySplit[0].ToLower())
  {
   case "termo":
    GetTermo(search);
    break;
   case "sufixo":
    GetSufixo(search + "%");
    break;
   case "prefixo":
    GetPrefixo(search + "%");
    break;
   case "prefsem":
    sem = GetSigsPrefixo("semantico", int.Parse(search));
    GetPrefixo("Semantico", int.Parse(search), sem);
    break;
   case "sufsem":
    sem = GetSigsSufixo("semantico", int.Parse(search));
    GetSufixo("Semantico", int.Parse(search), sem);
    break;
   case "prefgram":
    gram = GetSigsPrefixo("gramatical", int.Parse(search));
    GetPrefixo("gramatical", int.Parse(search), gram);
    break;
   case "sufgram":
    gram = GetSigsSufixo("gramatical", int.Parse(search));
    GetSufixo("gramatical", int.Parse(search), gram);
```

```
      break;
    case "lexema":
      GetLexema(search);
      break;
    case "slex":
      GetLexema(search + "%");
      break;
    case "idciencia":
      ciencia = GetCiencia(search);
      GetTermoCiencia(search, ciencia);
      break;
    default:
      LBError.Text = "Erro endereço mal formado";
      break;
  }
}
```

D3. Código da função C# de pesquisa por termos

```
/// <summary>
  /// Obter os termos abrangidos pela pesquisa
  /// </summary>
  /// <param name="search">Valor a pesquisar</param>
  private void GetTermo(string search)
  {
    LBSearchType.Text = null;
    //Url do módulo de visualização de termos
    string url = "~/Selects/ViewTermo.aspx?id=";
    //Criar a ligação ao SGDB
    SqlConnection con = new SqlConnection(connectionString);
    //String que contém o T-SQL a executar, obtém o id e o nome dos termos abrangidos pela pesquisa
    string sql = "Select IDTermo,Termo from Termo where Termo like @termo order by Termo asc";

    //Executar a operação T-SQL
    SqlCommand cmd = new SqlCommand(sql, con);
    int count = 0;
    int id = -1;
    try
    {
      //Passar os dados para os pârametros da query T-SQL
      cmd.Parameters.AddWithValue("@termo", search);
      //Abrir a ligação
      con.Open();
      SearchResults.Controls.Clear();
      //Ler os resultados obtidos pela query
      SqlDataReader reader = cmd.ExecuteReader();
      //Caso a pesquisa não devolva nenhum resultado, alargar a abrangência da mesma e executá-la novamente
      //Disponibilizar os resultados
      if (reader.HasRows)
      {
        LBSearchType.Text = "Pesquisa por: " + search;
        while (reader.Read())
        {
          DisplayLinks(reader.GetInt32(0), reader.GetString(1), url);
          id = reader.GetInt32(0);
          count++;
        }
      }

      else
      {
        LBSearchType.Text = "Sem resultados encontrados";
      }
      //Caso a pesquise encontre um só resultado, exibir imediatamente
      if (count == 1 && id != -1)
        Response.Redirect("~/Selects/ViewTermo.aspx?id=" + id);
      reader.Close();
    }
    catch (SqlException err)
    {
      //Capturar erros provenientes do servidor de base de dados
```

```
    LBSqlError.Text = "Erro Sql (GetTermo2): " + err.Message + ": " + err.LineNumber;
  }
  catch (Exception err)
  {
    //Capturar outros tipo de erro
    LBSqlError.Text = "Erro generico (GetTermo2): " + err.Message;
  }
  finally
  {
    //Fechar a ligação com o SQL Server
    con.Close();
  }
}
```

O preenchimento do controlo, um asp:Panel, que recolhe todos os termos abrangidos pela pesquisa, é realizado de forma dinâmica. A função que realiza esse preenchimento denomina-se *DisplayLinks* e realiza-o para qualquer tipo de pesquisa, seja por termo, prefixo, sufixo ou lexema.

```
/// <summary>
  /// Função que preenche o controlo dinâmico
  /// com o link respectivo seja ele um Termo,
  /// um Prefixo, um Lexema ou um Sufixo
  /// </summary>
  /// <param name="id">identificador único</param>
  /// <param name="value">nome</param>
  /// <param name="url">url para o módulo de visualização respectivo</param>
  private void DisplayLinks(int id, string value, string url) {
    HyperLink hl = new HyperLink();
    hl.ID = "hL" + id;
    hl.NavigateUrl = url + id;
    //hl.Target = "_blank";
    hl.CssClass = "LabelLink";
    hl.Text = value;
    SearchResults.Controls.Add(hl);
    SearchResults.Controls.Add(new LiteralControl("<br/>"));  }
```

Anexo E: Visualização da Informação

E1: Recolha de dados através de uma *query select*

O código seguinte é um exemplo de como é possível a definição de uma função C# para obter dados da base de dados, através da utilização de um *query* de *select* de uma tabela da base de dados.

```
/// <summary>
/// Obter os dados respectivos a uma dada ciência
/// </summary>
/// <param name="idCiencia">ID da Ciência a obter</param>
private void SelectCiencia(int idCiencia) {
  //Preencher o controlo dos subdomínios científicos pertencentes à ciência
  LBSubCat.DataBind();
  //Verificar se existem subdomínios científicos pertencentes à ciência
  if (LBSubCat.Items.Count == 0) BTN_SubCat.Visible = false;
  else BTN_SubCat.Visible = true;

  //Criar a ligação ao SGDB
  SqlConnection con = new SqlConnection(connectionString);
  //String que contém o T-SQL que obtém todos os dados relativos a uma ciência
  string sqlStr = "Select IDCiencia,Ciencia,Descricao,Notas from Ciencia where IDCiencia = @IDCienciaIn";
  SqlCommand cmd = new SqlCommand(sqlStr, con);

  try {
    //Passar os dados para os parâmetros da query T-SQL
    cmd.Parameters.AddWithValue("@IDCienciaIn", idCiencia);
    //Abrir a ligação
    con.Open();
    //Ler os resultados obtidos pela query
    SqlDataReader reader = cmd.ExecuteReader();
    if (reader.HasRows) {
      //Caso existam resultados preencher os controlos respectivos com os dados
      while (reader.Read()) {
        LBCiencia.Text = reader["Ciencia"].ToString();
        if (!reader.IsDBNull(2))
          TBDefinicao.Text = reader.GetString(2);
        else TBDefinicao.Text = null;
        if (!reader.IsDBNull(3))  {
          //Existem notas, exibi-las
          MVNotasCiencia.ActiveViewIndex = 0;
          TBNotasC.Text = "Notas: " + reader.GetString(3);
        }
        //Não existem notas
        else MVNotasCiencia.ActiveViewIndex = -1;
      }
    }
  }
  catch (SqlException err) {
    //Capturar erros provenientes do servidor de base de dados
    LBSqlError.Text = "LBSqlError: SelectCiencia: " + err.Message;
  }
```

```
    catch (Exception err) {
      //Capturar outros tipo de erro
      ErrorLB.Text = "ERROR: SelectCiencia: " + err.Message;
    }
    finally { //Fechar a ligação con.Close(); }  }
```

E2: Recolha de dados através de uma *View*

O código seguinte é um exemplo de como é possível a definição de uma função C# para obter dados da base de dados, através da utilização de um *query* de *select* para uma *view*.

```
/// <summary>
  /// Obter os dados de um prefixo, utilizando a view ViewPrefixo
  /// </summary>
  /// <param name="id">id do prefixo a obter</param>
  /// <returns>array de strings, onde irão ser armazenados os dados.
  /// A ordem dos parâmetros da view é seguida</returns>
  private string[] SelectPrefixo(int id) {
    string[] ret = null;
    //Criar a ligação ao SGDB
    SqlConnection con = new SqlConnection(connectionString);
    //String que contém o T-SQL a executar, utiliza a view como se ela fosse uma tabela normal
    string sql = "Select * from ViewPrefixo where IDPrefixo = @idPrefixoIn";
    try {
      //Executar a operação T-SQL
      SqlCommand cmd = new SqlCommand(sql, con);
      //Passar os dados para os parâmetros da query T-SQL
      cmd.Parameters.AddWithValue("@idPrefixoIn", id);
      con.Open();
      //Ler os resultados obtidos pela query
      SqlDataReader reader = cmd.ExecuteReader();
      //Caso haja resultados, guardá-los no array de strings
      if (reader.HasRows) {
        ret = new string[reader.FieldCount];
        while (reader.Read()) {
          for (int i = 0; i < reader.FieldCount; i++) {
            ret[i] = reader.GetValue(i).ToString();
          }
        }
      }
      else {
        //Alerta Caso o prefixo não exista
        LBError.Text = "Erro! Não existe Prefixo com esse id!";
      }
    }
    catch (SqlException err) {
      //Capturar erros provenientes do servidor de base de dados
      LBSqlError.Text = "Erro sql (SelectPrefixo): " + err.Message + ": " + err.LineNumber;
    }
    catch (Exception err) {
      //Capturar outros tipo de erro
      LBSqlError.Text = "Erro generico (SelectPrefixo): " + err.Message;
    }
    finally {
      //Fechar a ligação
      con.Close();
    }
    //Retornar o array de strings
    return ret;
  }
```

E3. Visualização dos constituintes morfológicos de um termo

A função C# apresentada a seguir, contém o código responsável pela disponibilização ordenada dos constituintes morfológicos de um termo.

```
/// <summary>
  /// Preenche o controlo asp:Label com os constituintes morfológicos
  /// respeitando a ordem armazenada
  /// </summary>
  /// <param name="idTermo">id do termo respectivo</param>
  /// <param name="formato">string que contém a ordem dos constituintes</param>
  private void DisplayFormato(int idTermo, string formato)
  {
    string urlLexema = "../Selects/ViewLexema.aspx?id=";
    string urlSufixo = "../Selects/ViewSufixo.aspx?id=";
    string urlPrefixo = "../Selects/ViewPrefixo.aspx?id=";
    LBTerminologia.Text = null;

    //Verifica se existe ordem armazenada
    if (!string.IsNullOrEmpty(formato))
    {
      //Separa a string ordem pelo caracter de separação ';'
      string[] termSplit = formato.Split(';');
      //Percorre cada constituinte
      for (int i = 0; i < termSplit.Length - 1; i++)
      {
        //Analisa a primeira letra da ordem
        switch (termSplit[i].Substring(0, 1))
        {
          //Lexema
          case "L":
            //Vai buscar o id do Lexema à tabela TermoSufixo que corresponde ao id o termo e a posicao
            int idLexema = GetIDLexema(idTermo, int.Parse(termSplit[i].Substring(1)));
            if (idLexema != -1)
            {
              //Vai buscar os dados do Lexema a partir do id obtido anteriormente
              string[] lexemaAll = GetAllLexema(idLexema);
              //Preenche os controlos com o lexema
              LBTerminologia.Text += "<a href='" + urlLexema + idLexema + "' class='LabelLink'>" +lexemaAll[1]+ "</a> ";

              if (lexemaAll[5].Equals("Grego") || lexemaAll[5].Equals("gr.") || lexemaAll[5].Equals("grego"))
                LBTerminologia.Text += lexemaAll[4] + " ";

              LBTerminologia.Text += lexemaAll[6].ToLower() + " " + lexemaAll[8] + " ";
            }
            else
              //Alerta de erro
              LBTerminologia.Text += " Erro ao obter o lexema ";
            break;
          //Sufixo
          case "S":
            //Vai buscar o id do Sufixo a tabela TermoSufixo que corresponde ao id o termo e a posicao
            int idSufixo = GetIDSufixo(idTermo, int.Parse(termSplit[i].Substring(1)));
            if (idSufixo != -1)
```

```
        {
          //Vai buscar os dados do sufixo a partir do id obtido anteriormente
          string[] sufixoAll = GetAllSufixo(idSufixo);
          //Preenche os controlos com o sufixo
          LBTerminologia.Text += "<a href='" + urlSufixo + idSufixo + "' class='LabelLink'>" + sufixoAll[1] + "</a> ";

          if (sufixoAll[7].Equals("Grego") || sufixoAll[7].Equals("gr.") || sufixoAll[7].Equals("grego"))
            LBTerminologia.Text += sufixoAll[4] + " ";

          LBTerminologia.Text += sufixoAll[8].ToLower() + " " + sufixoAll[2] + " ";
        }
        break;
      //Prefixo
      case "P":
        //Vai buscar o id do Prefixo à tabela TermoSufixo que corresponde ao id o termo e a posicao
        int idPrefixo = GetIDPrefixo(idTermo, int.Parse(termSplit[i].Substring(1)));
        if (idPrefixo != -1)
        {
          //Vai buscar os dados do prefixo a partir do id obtido anteriormente
          string[] prefixoAll = GetAllPrefixo(idPrefixo);
          //Preenche os controlos com o sufixo
          LBTerminologia.Text += "<a href='" + urlPrefixo + idPrefixo + "'class='LabelLink'>" + prefixoAll[1] + "</a> ";
          if (prefixoAll[7].Equals("Grego") || prefixoAll[7].Equals("gr.") || prefixoAll[7].Equals("grego"))
            LBTerminologia.Text += prefixoAll[4] + " ";
          LBTerminologia.Text += prefixoAll[8].ToLower() + " " + prefixoAll[2] + " ";
        }
        break;
    }
  }
 }
}
```

E4. Recolha do identificador único de um prefixo

A função C# aqui apresentada é utilizada pela função apresentada no Anexo E4, para recolher o identificador único de um prefixo. Ela serve de exemplo para os restantes constituintes morfológicos.

```
/// <summary>
  /// Recolhe o identificador único de um prefixo
  /// que está associado a um dado termo e a uma dada posição
  /// na Tabela TermoLexema
  /// </summary>
  /// <param name="idTermo">id do termo</param>
  /// <param name="posicao">posição do prefixo</param>
  /// <returns></returns>
  private int GetIDPrefixo(int idTermo, int posicao)
  {
    int ret = -1;
    //Criar a ligação ao SGDB
    SqlConnection con = new SqlConnection(connectionString);
    //String que contém o T-SQL a executar, obtém o id do prefixo
    string sql = "select refidprefixo from termoprefixo where refidtermo = @idtermo and posicao = @posicao";
    SqlCommand cmd = new SqlCommand(sql, con);
    try
    {
      //Passar os dados para os parâmetros da query T-SQL
      cmd.Parameters.AddWithValue("@idtermo", idTermo);
      cmd.Parameters.AddWithValue("@posicao", posicao);
      //Abrir a ligação
      con.Open();
      //guardar o id obtido
      ret = (int)cmd.ExecuteScalar();
    }
    catch (SqlException err)
    {
      //Capturar erros provenientes do servidor de base de dados
      LBSqlError.Text = "Erro sql (GetIDPrefixo): " + err.Message + ": " + err.LineNumber;
    }
    catch (Exception err)
    {
      //Capturar outros tipo de erro
      LBSqlError.Text = "Erro generico (GetIDPrefixo): " + err.Message;
    }
    finally
    {
      //Fechar a ligação
      con.Close();
    }
    //Retornar o id obtido pela query
    return ret;
  }
```

Anexo F: Funções Javascript

F1: Alerta Javascript

De seguida, é exibida a classe Alert.cs, responsável pela criação do alerta javascript.

```
using System.Web;
using System.Text;
using System.Web.UI;

/// <summary>
/// A JavaScript alert
/// </summary>
public static class Alert
{

    /// <summary>
    /// Exibe um alerta JavaScript (criado no lado o cliente) no browser
    /// </summary>
    /// <param name="message">A mensagem que é exiba no alerta.</param>
    public static void Show(string message)
    {
        //Limpa a mensagem para permitir o uso de plicas
        string cleanMessage = message.Replace("'", "\'");
        //O código javaScript do alerta
        string script = "<script type=text/javascript>alert('" + cleanMessage + "');</script>";

        //Vai buscar a página que está a ser executada
        Page page = HttpContext.Current.CurrentHandler as Page;

        //Verifica se é uma página e se o script ainda não está colocado na página
        if (page != null && !page.ClientScript.IsClientScriptBlockRegistered("alert"))
        {
            //Regista o script na página
            page.ClientScript.RegisterStartupScript(typeof(Alert), "alert", script);
        }
    }
}
```

F2: Conversão para caracteres gregos

É apresentado um excerto da função javascript responsável pela conversão dos caracteres do teclado para os caracteres gregos.

```
function greeklish(x)
 {
 var y=document.getElementById(x).value;
y=y.replace('``','“');
y=y.replace('\\"','”');
y=y.replace('´´','”');
y=y.replace(',,',',„');
y=y.replace('<<','«');
y=y.replace('>>','»');
y=y.replace(':',',·');
y=y.replace('?',';');
y=y.replace(' )',')');
y=y.replace('a','α');
y=y.replace('α/','ά');
y=y.replace('α\\','ὰ');
y=y.replace('α=','ᾶ');
y=y.replace('α)','ἀ');
y=y.replace('α(','ἁ');
y=y.replace('α_','ᾱ');
y=y.replace('α-','ᾰ');
```

```
y=y.replace('α|','ᾳ');
y=y.replace('ᾳ\\','ᾲ');
y=y.replace('ᾳ/','ᾴ');
y=y.replace('ᾳ\\','ᾲ');
y=y.replace('ᾳ/','ᾴ');
y=y.replace('ᾳ=','ᾷ');
y=y.replace('ᾱ=','ᾷ');
y=y.replace('ᾳ|','ᾴ');
y=y.replace('ᾳ|','ᾴ');
y=y.replace('ᾳ|','ᾴ');
y=y.replace('ᾳ|','ᾴ');
y=y.replace('ᾳ|','ᾴ');
y=y.replace('ᾳ|','ᾴ');
y=y.replace('ᾳ|','ᾴ');
y=y.replace('ᾳ|','ᾴ');
y=y.replace('ᾳ|','ᾴ');
y=y.replace('ᾳ|','ᾴ');
(...)
}
```

Anexo H: Inserção da Informação

H1: Inserção através de um controlo asp:SqlDataSource

O código aspx que define a inserção através de um controlo asp:SqlDataSource é o seguinte:

```
<asp:SqlDataSource ID="InsertCiencia" runat="server" ConnectionString="<%$ ConnectionStrings:DiciterV5 %>"
   InsertCommand="INSERT INTO [Ciencia] ([Ciencia], [Descricao], [notas]) VALUES (@cienciaIn,
@descricaoIn, @notasIn)"
   SelectCommand="SELECT [IDCiencia], [Ciencia], [Descricao], [notas] FROM [Ciencia]">
   <InsertParameters>
    <asp:ControlParameter ControlID="CienciaTB" Name="cienciaIn" PropertyName="Text" />
    <asp:ControlParameter ControlID="TBDescricao" Name="descricaoIn" PropertyName="Text" />
    <asp:ControlParameter ControlID="NotasTB" Name="notasIn" PropertyName="Text" />
   </InsertParameters>
 </asp:SqlDataSource>
```

A função que invoca este controlo para inserção dos dados é a seguinte:

```
/// <summary>
  /// Função associada ao evento click do botão submeter que
  /// insere os dados da ciência na base dados
  /// </summary>
  /// <param name="sender"></param>
  /// <param name="e"></param>
  protected void Submit_Click(object sender, EventArgs e)
  {
    LBSqlError.Text = "";
    int insertedRows = 0;
    try
    {
      //Invocar a operação de inserção configurada no controlo asp:SqlDataSource
      insertedRows = InsertCiencia.Insert();
      //Verificar se a inserção foi realizada.
      if (insertedRows != 0)
      {
        //Lançar um alerta de sucesso
        Alert.Show("Inserção da ciencia " + CienciaTB.Text + "foi efectuada!");
        CienciaTB.Enabled = false;
        TBDescricao.Enabled = false;
        NotasTB.Enabled = false;
        Submit.Enabled = false;
        Clear.Text = "Nova";
      }
    }
    catch (SqlException err)
    {
      //Capturar a excepção gerada pelo índice único configurado na tabela
```

```
    if (err.ErrorCode == -2146232060)
    {
      Alert.Show("A ciencia já existe!");
      LBSqlError.Text = "A ciencia já " + CienciaTB.Text + "existe!";
      CienciaTB.Enabled = false;
      TBDescricao.Enabled = false;
      NotasTB.Enabled = false;
      Submit.Enabled = false;
      Clear.Text = "Nova";
    }
    else
      //Capturar outro tipo de erro gerado pelo SQL Server
      LBSqlError.Text = "Sql Error: " + err.Message + err.LineNumber;
  }
  catch (Exception err)
  {
    //Capurar outros erros
    LBSqlError.Text = "Generic Error: " + err.Message;
  }
}
```

H2: Inserção através de um *stored procedure*

A função seguinte exemplifica como a inserção dos dados pode ser feita através da utilização de um *stored procedure* definido no sistema de gestão de base de dados:

```
/// <summary>
/// Inserir Prefixo na base de dados
/// </summary>
/// <param name="prefixo">nome</param>
/// <param name="definicao">definição</param>
/// <param name="notas">notas</param>
/// <param name="nIDLingua">id da língua</param>
/// <param name="nIDSigSem">id do significado semântico</param>
/// <param name="nIDSigGram">id do significado gramatical</param>
/// <returns>o id do elemento inserido</returns>
public static int InsertPrefixo(string prefixo,string definicao,string notas,int nIDLingua,int nIDSigSem,int nIDSigGram )
 {
  int ret = -1;
  //Url do módulo de visualização de termos
  SqlConnection con = new SqlConnection(connectionString);
  SqlCommand cmd = new SqlCommand();

  try {
   cmd.Connection = con;
   //Indicar a utilização de um stored procedure
   cmd.CommandType = CommandType.StoredProcedure;
   //Indicar o nome do stored procedure
   cmd.CommandText = "InsertPrefixo";
   //Passar os dados para os parâmetros do stored procedure
   cmd.Parameters.AddWithValue("@prefixoIn", prefixo);
   cmd.Parameters.AddWithValue("@definicaoIn", definicao);
   cmd.Parameters.AddWithValue("@refIDlinguaIn", nIDLingua);
   cmd.Parameters.AddWithValue("@notasIn", notas);
   cmd.Parameters.AddWithValue("@refIDSsemIn", nIDSigSem);
   cmd.Parameters.AddWithValue("@refIDSgramIn", nIDSigGram);
   //Parâmetro de retorno que contém o id do elemento inserido
   cmd.Parameters.Add(new SqlParameter("@ret", SqlDbType.Int));
   cmd.Parameters["@ret"].Direction = ParameterDirection.ReturnValue;

   //Abrir a ligação
   con.Open();
   //Iniciar uma transacção
   SqlTransaction tran = con.BeginTransaction();
   cmd.Transaction = tran;
   cmd.ExecuteNonQuery();
   //obter o id do elemento inserido
   ret = (int)cmd.Parameters["@ret"].Value;
   //Realizar o commit da transacção
   tran.Commit();
  }
  catch (SqlException err) { (...) }
  //Retornar o id inserido
  return ret;
 }
```

Anexo I: *WebServices*

I1: Classe WebServices.cs

Nesta classe estão presentes todas as funções responsáveis pelo funcionamento do controlo ASP .NET AJAX *AutoCompleteExtender*. Como exemplo é apresentado o código da função responsável pela recolha da informação da Língua. Para as restantes funções a implementação é idêntica, mudando naturalmente as tabelas de onde a informação é recolhida. De realçar o aparecimento, antes de cada função, da palavra *[WebMethod]* que marca as funções como pertencentes a um *WebService*.

```
using System;
using System.Collections;
using System.Linq;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Xml.Linq;
using System.Web.Configuration;
using System.Data;
using System.Data.SqlClient;

/// <summary>
/// Classe que contém a lógica de funcionamento dos WebServices
/// Contém todas as funções responsáveis para o controlo ASP .NET AJAX AutoCompleteExtender.
/// </summary>
[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
// To allow this Web Service to be called from script, using ASP.NET AJAX, uncomment the following line.
[System.Web.Script.Services.ScriptService]
public class WebService : System.Web.Services.WebService
{
  private string connectionString = WebConfigurationManager.ConnectionStrings["DiciterV5"].ConnectionString;
  public WebService()
  {

    //Uncomment the following line if using designed components
    //InitializeComponent();
  }
  /// <summary>
  /// Função que contém o código para o preenchimento automático de caixas de textos para a língua
  /// </summary>
  /// <param name="prefixText">valor a pesquisar</param>
  /// <returns></returns>
  [WebMethod]
  public string[] GetLingua(string prefixText)
  {
    //String que contém o T-SQL a executar
    string sql = "Select * from Lingua Where Lingua like @prefixText";
```

```
  //Criar a ligação e executar a query
  //Permite obtenção dos dados para depois serem armazenados numa variável do tipo DataTable
  SqlDataAdapter da = new SqlDataAdapter(sql, connectionString);
  //Passar os dados para os parâmetros da query T-SQL
  da.SelectCommand.Parameters.Add("@prefixText", SqlDbType.NVarChar, 50).Value = prefixText + "%";
  //Uma tabela para armazenar os dados obtidos
  DataTable dt = new DataTable();
  da.Fill(dt);
  //Variável para armazenar os dados obtidos
  string[] items = new string[dt.Rows.Count];
  int i = 0;
  //Percorre a DataTable e armazena os dados na variável criar antes
  foreach (DataRow dr in dt.Rows)
  {
    items.SetValue(dr["Lingua"].ToString(), i);
    i++;
  }
  //devolver os dados obtidos pela pesquisa
  return items;
}

[WebMethod]
public string[] GetClasse(string prefixText) (...)
[WebMethod]
public string[] GetCiencia(string prefixText) ) (…)
[WebMethod]
public string[] GetLexema(string prefixText) (…)
[WebMethod]
public string[] GetPrefixo(string prefixText) (…)
[WebMethod]
public string[] GetSufixo(string prefixText) (…)
[WebMethod]
public string[] GetSigSemSuf(string prefixText) (…)
 [WebMethod]
public string[] GetSigGramSuf(string prefixText) (…)
  [WebMethod]
public string[] GetSigSemPref(string prefixText) (…)
 [WebMethod]
public string[] GetSigGramPref(string prefixText) (…)
}
```

Anexo J: Aspectos Particulares do Módulo InsertTermo.aspx

J1: Mecanismos de suporte a várias definições

O código seguinte é responsável pela construção dinâmica dos controlos necessários para o suporte de definições distintas, com classe gramaticais diferentes, se necessário:

```
/// <summary>
/// Criar as caixas de texto dependendo do número indicado no controlo TBDefNum
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void TBDefNum_Changed(object sender, EventArgs e)
{
  //Converte o valor da caixa de texto para inteiro
  int num = int.Parse(TBDefNum.Text);
  //Guarda o valor numa variável de sessão para utilização futura
  Session["NumDef"] = num;
  switch (num)
  {
    //Se for 0 não cria nenhuma
    case 0: break;
    //Só uma definição uma caixa de texto
    case 1:
      PanelDef2.Controls.Clear();
      TBClasseList.Clear();
      TBDefinicaoList.Clear();
      //Criar as caixas de texto
      TextBox TBDef = new TextBox();
      TBDef.ID = "TBDef" + TBDefinicaoList.Count;
      TBDef.CssClass = "textbox";
      TBDef.TextMode = TextBoxMode.MultiLine;
      TBDef.Width = Unit.Parse("95%");
      TBDef.Height = Unit.Parse("70px");
      //Associar uma estrutura e as caixas de texto criadas anteriormente
      PanelDef2.Controls.Add(new LiteralControl("<tr>"));
      PanelDef2.Controls.Add(new LiteralControl("<td class='tablecell'> Definição:"));
      PanelDef2.Controls.Add(new LiteralControl("</td>"));
      PanelDef2.Controls.Add(new LiteralControl("<td colspan='3'>"));
      PanelDef2.Controls.Add(TBDef);
      PanelDef2.Controls.Add(new LiteralControl("</td>"));
      PanelDef2.Controls.Add(new LiteralControl("</tr>"));
      TBDefinicaoList.Add(TBDef);
      break;
    //mais do que uma definição
    default:
      PanelDef2.Controls.Clear();
      TBClasseList.Clear();
      TBDefinicaoList.Clear();
      //Repetir quantas vezes quanto o número de definições
      for (int i = 0; i < num; i++)
      {
        //Associar uma estrutura e as caixas de textos criadas anteriormente ao painel
```

```
        PanelDef2.Controls.Add(new LiteralControl("<tr>"));
        PanelDef2.Controls.Add(new LiteralControl("<td colspan='3' class='tableHeader' style='text; font-style: italic;'>"));
        PanelDef2.Controls.Add(new LiteralControl((i + 1).ToString() + "ª definição:"));
        PanelDef2.Controls.Add(new LiteralControl("</td>"));
        PanelDef2.Controls.Add(new LiteralControl("</tr>"));
        PanelDef2.Controls.Add(new LiteralControl("<tr>"));
        PanelDef2.Controls.Add(new LiteralControl("<td class='tablecell'>Classe:"));
        PanelDef2.Controls.Add(new LiteralControl("</td>"));
        PanelDef2.Controls.Add(new LiteralControl("<td>"));
        //Criar as caixas de texto
        TextBox TBClasseDim1 = new TextBox();
        TBClasseDim1.ID = "TBClasse" + i;
        TBClasseDim1.CssClass = "textbox";
        PanelDef2.Controls.Add(TBClasseDim1);
        PanelDef2.Controls.Add(new LiteralControl("    <a style='font-style: italic; font-size: 11px;
color:#444444;font-family: Tahoma,Helvetica,Helv;'>opcional</a>"));
        PanelDef2.Controls.Add(new LiteralControl("</td>"));
        PanelDef2.Controls.Add(new LiteralControl("</tr>"));
        //Associar a caixa de texto para a classe
        TBClasseList.Add(TBClasseDim1);
        PanelDef2.Controls.Add(new LiteralControl("<tr>"));
        PanelDef2.Controls.Add(new LiteralControl("<td class='tablecell'> Definição:"));
        PanelDef2.Controls.Add(new LiteralControl("</td>"));
        PanelDef2.Controls.Add(new LiteralControl("<td colspan='3'>"));
        TextBox TBDef1 = new TextBox();
        TBDef1.ID = "TBDef" + i;
        TBDef1.CssClass = "textbox";
        TBDef1.Width = Unit.Parse("95%");
        TBDef1.Height = Unit.Parse("70px");
        TBDef1.TextMode = TextBoxMode.MultiLine;
        //Associar a caixa de texto para as definições
        PanelDef2.Controls.Add(TBDef1);
        PanelDef2.Controls.Add(new LiteralControl("</td>"));
        PanelDef2.Controls.Add(new LiteralControl("</tr>"));
        TBDefinicaoList.Add(TBDef1);
      }
      break;
    }
  }
```

O código de reconstrução destas caixas de texto a seguir a um evento de *postback* da página é em tudo idêntico ao apresentado em cima.

A concatenação numa estrutura pré-definida de todas as definições especificadas, forma a poderem ser armazenadas e recuperadas quando necessário é realizado pela função seguinte:

```
/// <summary>
/// Criar a estrura da definição
/// </summary>
/// <returns>devolve as definições concatenadas</returns>
private string SetDefinition()
{
  string def = null;
  //Número de definições
```

```
    int countNumDef = int.Parse(Session["NumDef"].ToString());

    switch (countNumDef)
    {
      //0 definições
      case 0:
        break;
      //1 definição
      case 1:
        def = TBDefinicaoList[0].Text;
        break;
      //Várias definições
      default:
        //Percorrer todas as caixas de textos criadas dinaminamente
        for (int i = 0; i < countNumDef; i++)
        {
          //concatenação das classe
s (caso existam) e das definições seguindo a estrutura definida
          if (!string.IsNullOrEmpty(TBClasseList[i].Text))
            def += (i + 1) + ". " + "(" + TBClasseList[i].Text + ") " + TBDefinicaoList[i].Text + "; " + "<br/>";
          else
            def += (i + 1) + ". " + TBDefinicaoList[i].Text + "; " + "<br/>";
        }
        break;
    }
    //Retornar a concatenção obtida
    return def;
  }
```